8-1-2009

GPU-based implementation of real-time system for spiking neural networks

Dmitri Yudanov

GPU-Based Implementation of Real-Time System for Spiking Neural Networks

by

Dmitri Yudanov

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Engineering

Supervised by

Dr. Muhammad Shaaban
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, NY
August 2009

Approved By:

Dr. Muhammad Shaaban

Primary Advisor – R.I.T. Dept. of Computer Engineering

Dr. Roy Melton

Secondary Advisor – R.I.T. Dept. of Computer Engineering

Dr. Leon Reznik

Secondary Advisor – R.I.T. Dept. of Computer Science

Thesis Release Permission Form

Rochester Institute of Technology
Kate Gleason College of Engineering

Title: GPU-Based Implementation of Real-Time System for Spiking Neural Networks

I, Dmitri Yudanov, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or part.

_____

Dmitri Yudanov

_____

Date

Dedication

To my Mom and Dad.

Acknowledgements

Big “Thank You” to Dr. Muhammad Shaaban, Dr. Roy Metlon, Dr. Leon Reznik, Dr. Robert Bowman, Dr. Dhireesha Kudithipudi, Dr. Tejasvi Das, Dr. Juan Cockburn, Dr. Marcin Lukowiak, Dr. Andres Kwasinski, Jesse Harvey, Michael Romero, Jeremy Espenshade, Yulia Desyatkova, Richard Tolleson.

Abstract

Real-time simulations of biological neural networks (BNNs) provide a natural platform for applications in a variety of fields: data classification and pattern recognition, prediction and estimation, signal processing, control and robotics, prosthetics, neurological and neuroscientific modeling. BNNs possess inherently parallel architecture and operate in continuous signal domain. Spiking neural networks (SNNs) are type of BNNs with reduced signal dynamic range: communication between neurons occurs by means of time-stamped events (spikes). SNNs allow reduction of algorithmic complexity and communication data size at a price of little loss in accuracy. Simulation of SNNs using traditional sequential computer architectures results in significant time penalty. This penalty prohibits application of SNNs in real-time systems.

Graphical processing units (GPUs) are cost effective devices specifically designed to exploit parallel shared memory-based floating point operations applied not only to computer graphics, but also to scientific computations. This makes them an attractive solution for SNN simulation compared to that of FPGA, ASIC and cluster message passing computing systems. Successful implementations of GPU-based SNN simulations have been already reported.

The contribution of this thesis is the development of a scalable GPU-based real-time system that provides initial framework for design and application of SNNs in various domains. The system delivers an interface that establishes communication with neurons in the network as well as visualizes the outcome produced by the network. Accuracy of the simulation is emphasized due to its importance in the systems that exploit spike time dependent plasticity, classical conditioning and learning. As a result, a small network of 3840 Izhikevich neurons implemented as a hybrid system with Parker-Sochacki numerical integration method achieves real time operation on GTX260 device. An application case study of the system modeling receptor layer of retina is reviewed.

Table of Contents

List of Figures

List of Tables

Glossary

ANN	Artificial neural network
BNN	Biological neural network
CUDA	Compute unified device architecture
GPU	Graphical processing unit
IZ	Izhikevich neuron model
LTD	Long term depression
LTP	Long term potentiation
PS	Parker-Sochacki numerical integration technique
SNN	Spiking neural network
STDP	Spike time dependent plasticity

Chapter 1 Introduction

A spiking neural network is a model of a real biological neural network. A building block of an SNN is a mathematical model of a neuron. The choice of this model is design-specific. Neurons communicate with each other by spike events, modeled as time-stamped pulses. A choice of computational system used of SNN modeling defines accuracy of the entire network. There are synchronous systems that tie events to a time grid of simulation flow. Precision of the grid is defined by the magnitude of a time step: the smaller it is the better the precision and accuracy, but more steps per unit of simulation time have to be computed. There are asynchronous systems that update model variables at the time of incoming event and thus, model event time precisely. Two identical SNNs, excited with identical stimuli, but implemented as synchronous and asynchronous systems, do not produce the same spiking pattern unless a time step in synchronous system implementation is extremely small [1], [2].

There are biological mechanisms that require precise timing, for example, spike time dependent plasticity, which is one of the most common research subjects in neuroscience. STDP simulation with system-intrinsic quantization error, introduced by a synchronous system, results in incorrect evolution of network topology due to inability of the system to distinguish between long term potentiation and long term depression [1].

Recently introduced in computational neuroscience Parker-Sochacki numerical integration method [2], applied to biologically plausible phenomenological neuron model developed by Izhikevich [3], provides accuracy that is suitable for precise simulation of SNNs with biological mechanisms that require exact timing. In fact, it was shown that such simulations exhaust full double precision error tolerance requirement [2].

This work brings PS model of IZ-based SNNs into the world of parallel simulations based on CUDA GPUs. Another goal of this work is to provide a suitable interface to the SNN for its application in various domains.

The text is organized in the following way. Chapter two introduces the reader to neural networks. The first section reviews basic biophysical processes transpiring in biological neurons, such as action potential, synaptic transmission, and neuronal integration. The last part of the section introduces and provides an explanation to the processes requiring precise timing such as spike time dependent plasticity, long term potentiation, long term depression, and classical conditioning. Section two concentrates on modeling aspects of some processes reviewed in the first section, and introduces three most commonly used models of membrane dynamics as well as a simple model of synaptic transmission commonly utilized in SNNs. Section three briefly defines, contrasts, and compares spiking and artificial neural networks. Finally, the last section presents several application examples of SNNs.

The goal of the third chapter is to present modeling approaches to SNNs from the system perspective. The accent is made on time-related aspects of the system modeling: synchronous vs. event driven in the light of two major computational parts of SNNs: update and propagation phases. Suitability of system types for modeling precise time processes, defined in the second chapter, is analyzed. The chapter also provides the insight into important system components such as numerical integration techniques utilized in software and digital hardware models, as well as synaptic data structures responsible for network topology storage.

Chapter four reviews major implementation strategies applicable to SNNs and provides some examples: integrated circuits, programmable logic, and parallel software. Parallel software section is concentrated on GPU implementation, namely CUDA-enabled GPUs. CUDA architecture and its major optimization strategies are reviewed. Examples of works implementing SNNs on GPUs are analyzed.

Chapter five provides the insight into design and implementation of this project as well as justifications behind the design choice and the strategy selected. The chapter reviews these aspects from the perspective of update and propagation computational phases defined in Chapter three as well as it describes the interface to the system.

Chapter six concentrates on the results and analysis of the system defined in Chapter five from the perspective of execution time, scalability and interfacing with SNN.

Finally, the last chapter draws overall conclusions and proposes the future work.

Chapter 2 Introduction to Neural Networks

2.1. Essential neuroscience

This section reviews basic concepts of neuroscience and introduces definitions that will be used throughout the text. It defines biological basis for neuroscientific modeling.

2.1.1 Membrane Dynamics

A neuron is a building block of nervous system. Morphological structure of a neuron is depicted in Figure 2.1-1.

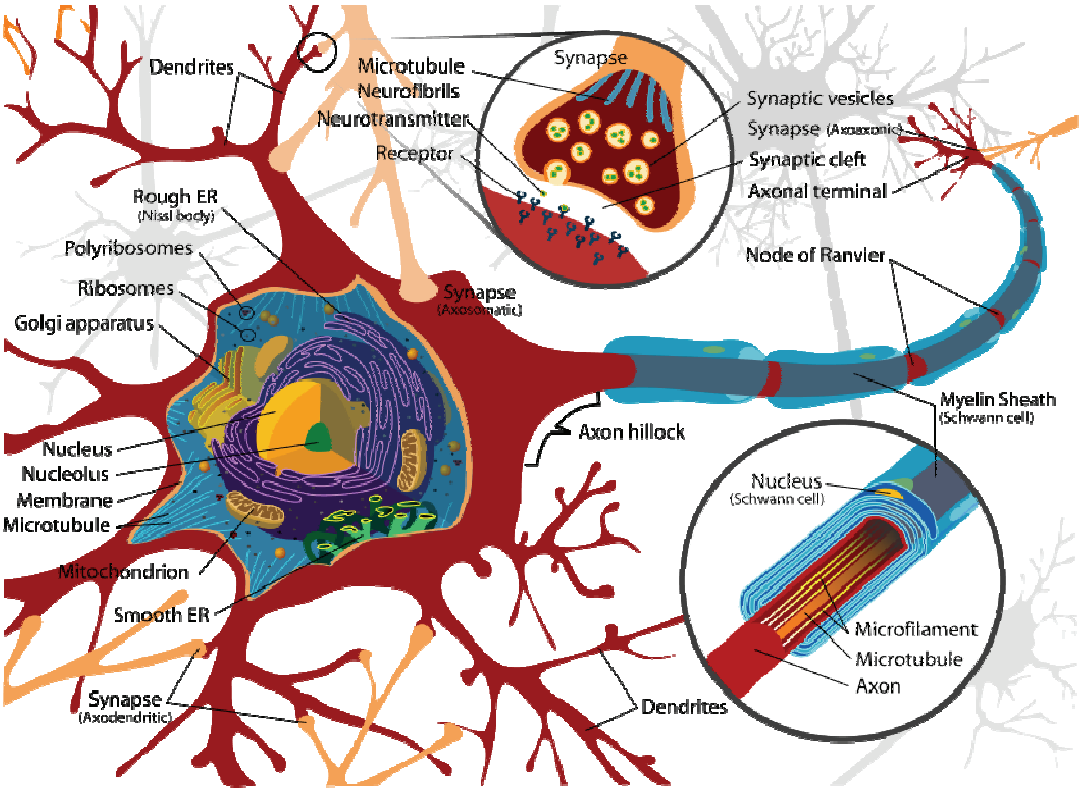


Figure 2.1-1 Morphological structure of a neuron

From engineering perspective the main functional element of a neuron is its membrane. Made of a lipid bilayer, the membrane has high resistance (1 TOhm for the surface of a typical spinal motor neuron [4]) and therefore, plays a role of the insulator. Placed in aqueous solution it separates extracellular space (outside the cell) from cytoplasmic space (inside the cell) in respect to a nerve cell that it is a part of. The membrane also plays a role of the capacitor (typically 1 $\mu F/cm^2$ [4]) because the aqueous solution where it resides contains various ion species such as Na^+, K^+, Ca^{2+}, Mg^{2+}, Cl^-, and organic anions A^-. Concentration of these ion species is different in extracellular and cytoplasmic sides, which results in separation of charges and makes membrane capacitance charged so that the potential across the membrane is between -60 mV and -70 mV relative to extracellular side [4]. This potential is called resting potential with emphasis that no electrical disturbances occur across the membrane. However, the membrane is note opaque because of the presence of electronic devices on its surface called ion channels (Figure 2.1-2).

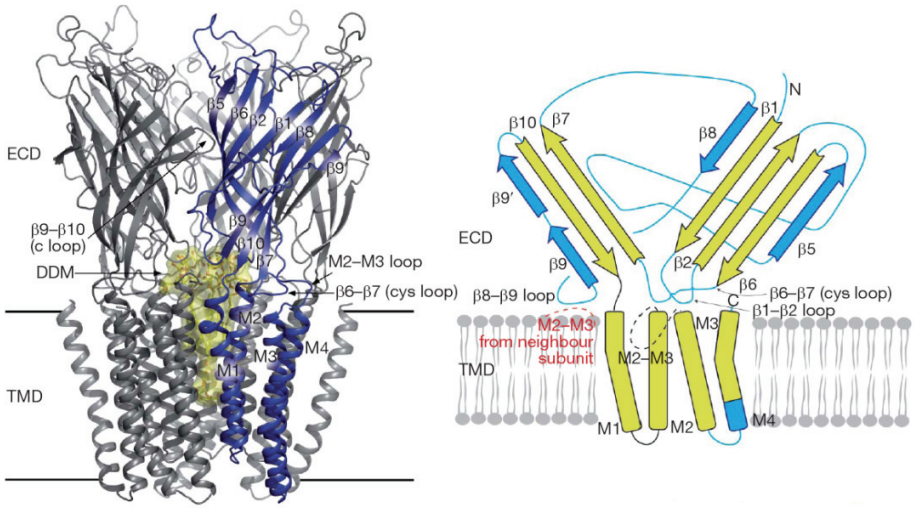


Figure 2.1-2 X-ray structure of *Gloeobacter violaceus* pentameric ligand-gated ion channel [5]

Ion channels provide openings in the membrane under various conditions and permit a variety of ions to cross the membrane surface. Ion channels can be classified as ligand-gated (opening or closing occurs in response to chemical binding at designated site

of ion channel), voltage-gated (opening or closing occurs in response to the potential), mechanically-gated (opening or closing occurs in response to pressure or stretch) and resting (these ion channels normally stay open at rest). Ion channels also can be classified in respect to the ions that they permit to pass through the membrane surface (selective permeability).

At rest Na^+, $K^+$and Cl^- ion channels are open, which decreases membrane resistance approximately by factor of 40,000. Thus, membrane capacitance (C_m) leaks charge. However, another kind of electronic devices, ion pumps (Figure 2.1-3), prevent the membrane capacitor from discharging and maintain a steady resting potential because of active ion transport that they generate: Na^+ is pumped to the extracellular side, K^+ is pumped to the intracellular side.

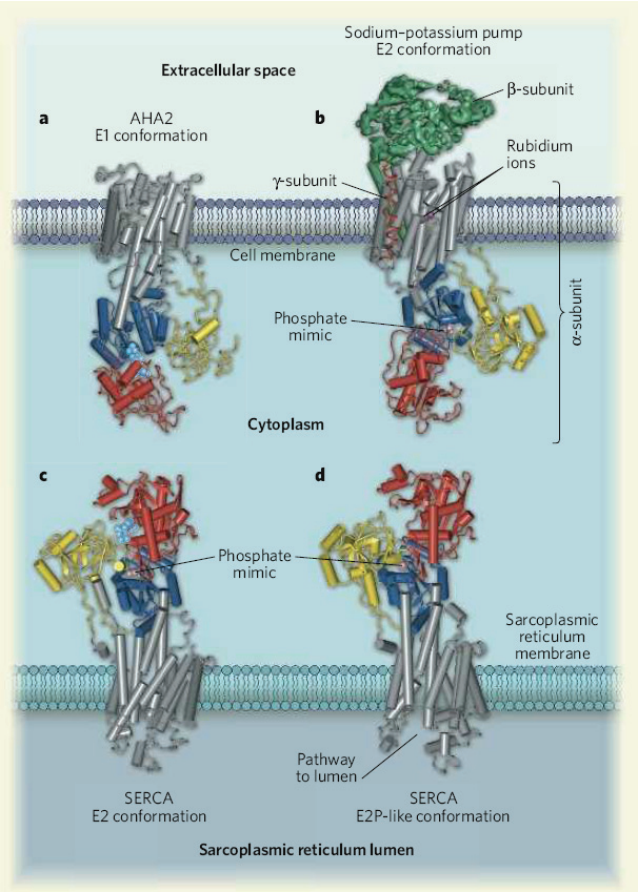


Figure 2.1-3 P-type ion pumps transport ions across either cell membranes (a,b) or membranes of intracellular organelles such as the sarcoplasmic reticulum(c,d) [6]

With each cycle the pump extrudes three Na^+ ions and brings in two K^+ ions, thus producing net outward current (electrogenic property of the pump). Therefore, the pump creates the force that tends to lower membrane potential. However, existence of resting channels results in a passive resistive path for the ions and establishes resting potential as an equilibrium point.

As a consequence, there are two forces that impact ion dynamics of the membrane:

1) Chemical driving force on each individual ion that depends on concentration gradient of that ion species across the membrane. This force is controlled by density of resting ion channels on the membrane surface and ion pump activity. The potential between ions of the same type that is established due to concentration gradient force, assuming that all other ion species are absent, is called equilibrium or Nernst potential.

2) Electrical driving force defined by the potential differences across the membrane as a result of membrane capacitor charge. This force depends on the total charge separated by membrane, which depends on concentration gradient of each ion type on both sides of the membrane and membrane surface area. Consequently, both forces are interdependent.

At its steady state forces and concentration gradients are balanced, net ion flux across the membrane is zero, and its resting potential is about -60 mV (Table 2.1-1).

Ion species	**Concentration in cytoplasm,** $[X]_i$	**Concentration in extracellular fluid,** $[X]_0$	**Nernst Potential,** $E_x = \frac{RT}{zF}\ln\frac{[X]_0}{[X]_i}$
	(mM)	**(mM)**	**(mV)**
K^+	400	20	-75
Na^+	50	440	+55
Cl^-	52	560	-60

Table 2.1-1Distribution of the major ions across a neuronal membrane at rest: the giant axon of the squid. Nernst potential in reference to extracellular side (chosen as ground by convention) [4]

where R is gas constant, T is temperature (K), z is ion valence, F is Faraday constant.

The system at its steady state can be expressed as an electrical circuit (Figure 2.1-4).

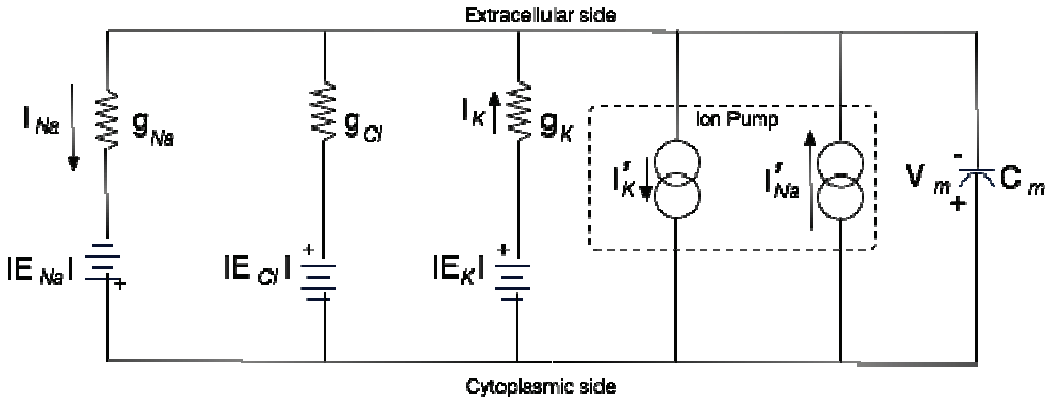


Figure 2.1-4 Circuit representation of neuron membrane at rest

In this circuit it is assumed that equilibrium potential of ion species doesn't change if resistive load is applied to the species source, and therefore, the source can be modeled as a voltage source or a battery with voltage $E_{ion\ type}$ across it. Polarity of this battery is defined by concentration gradient force relative to the extracellular side and ion species carrier charge. Resting channels are modeled as conductances $g_{ion\ type}$ that provide paths for steady state currents $I_{ion\ type}$. Currents due to ion pump are shown as $I'_{ion\ type}$, and polarized membrane capacitance is C_m. I_{Cl} is not shown because its value is zero since its Nernst potential is equal to the resting potential, and therefore, there is no net force that drives Cl^- ions in or out of the cell. An additional conductance path for other possible ions that model leakage current I_{leak} is omitted for simplicity. As a result, resting potential can be represented by the following equation [4]:

$$V_m = \frac{I_{total}}{g_{total}} = \frac{E_{Na}g_{Na} + E_K g_K + E_{Cl}g_{Cl}}{g_{Na} + g_K + g_{Cl}} \tag{2.1-1}$$

At its steady state the system balances ion flux and establishes equilibrium point at resting potential. Na^+ ions are pumped to the extracellular side and leak back through g_{Na} due to chemical driving force. However, their concentration is balanced in such a

way that there are more Na^+ ions on the extracellular side. K^+ ions are pumped to the cytoplasmic side and leak away from the cell through g_K again due to chemical driving force. Their concentration is balanced so that there are more K^+ ions in cytoplasmic side. As a rule, at rest $g_{Na} < g_K$. Thus, electrical and chemical forces affecting Na^+ ions are larger than those of K^+ ions. Besides, electrogenic pump property generates more Na^+ current than K^+ current. This Na^+ current balances with equivalent Na^+ current through Na^+ resting channels. As a result, the overall balance point (resting potential) is shifted towards the negative side since Na^+ current is directed inward.

Artificially injected into a cell current I_{inj} disturbs the balance and moves the potential in either direction: up if I_{inj} is inward, and down if I_{inj} is outward. Because net current ($I_{inj}+I_{Na} + I_K + I_{leak} + I_{Cl} + I'_{pump}$) follows I_{inj}, resting potential is said to be reversal potential, since the current changes its direction about its value. Reversal potential can be viewed as a voltage source in Thevenin equivalent circuit, which can be obtained from the circuit in Figure 2.1-4.

If membrane is disturbed electrically due to artificially injected current, so that net current is inward and V_m increases (depolarization), the system changes its behavior. Depolarization may result in membrane potential reaching threshold potential of voltage-gated Na^+ channels. In order to reach this threshold the membrane capacitance has to accumulate some charge, which results in instant capacitive current (Figure 2.1-5, I_C). If this threshold is reached, ion channels start opening rapidly providing influx of Na^+ ions into the cell.

During depolarization voltage-gated Na^+ channels open stochastically. However, when combined they produce aggregated exponential current influx (Figure 2.1-5, I_{Na}). Their stochastic behavior is defined by their density on the surface of the membrane. The density is the highest at the part of a cell called axon hillock (Figure 2.1-1).

Rapid opening of voltage-gated Na^+ channels results in exponential escape of membrane potential toward equilibrium potential of Na^+ ions (Figure 2.1-6). However, membrane potential doesn't reach that point because of passive K^+ ion efflux through resting channels (Figure 2.1-5, I_l). Besides, voltage-gated Na^+ channels start to close (inactivation process) and voltage-gated K^+ channels gradually continue to open. As a

consequence, net current into the cell changes its direction from inward to outward (Figure 2.1-5, right trace) and the membrane potential returns to its resting potential value (repolarization). As a rule, the membrane potential falls slightly below its resting potential value (hyperpolarization) because of prolonged opening of K^+ channels.

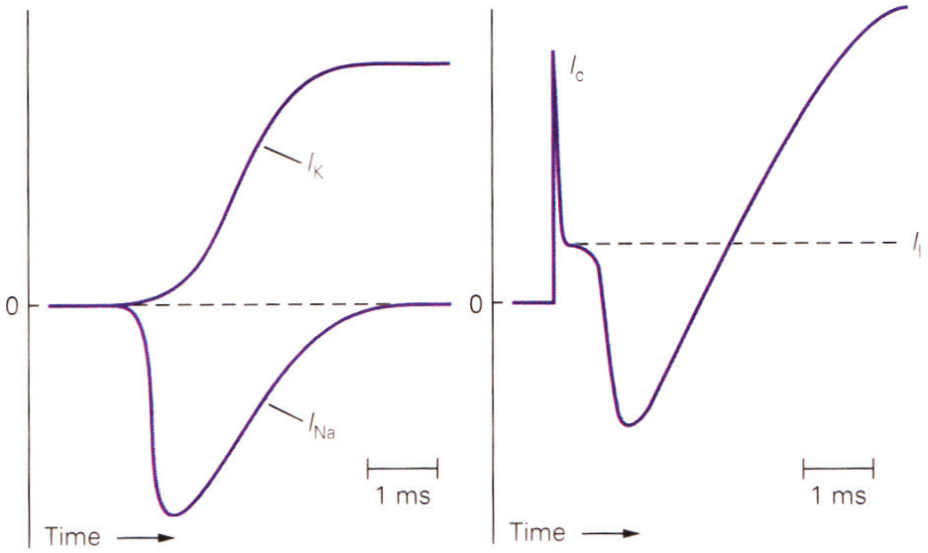


Figure 2.1-5 Ion currents during action potential [4]

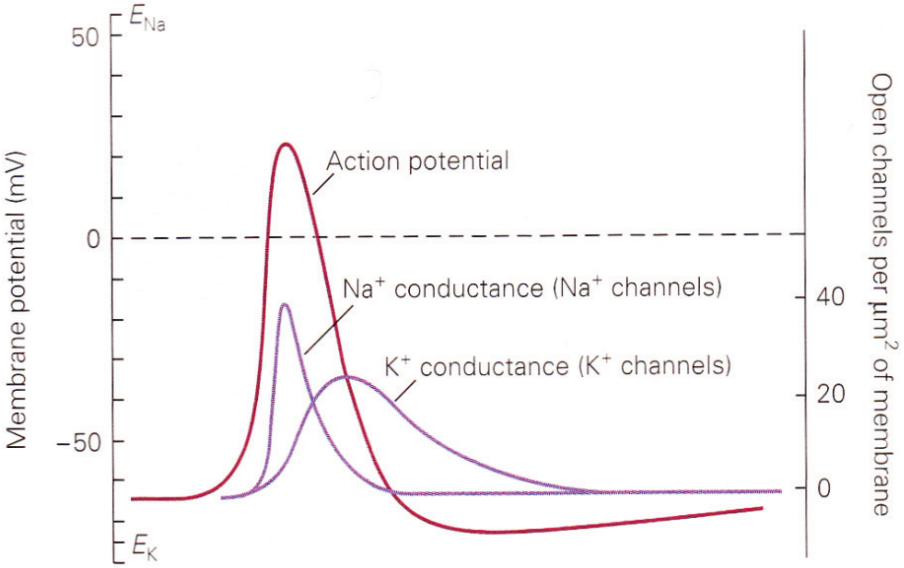


Figure 2.1-6 V_m, g_{Na} and g_K during action potential [4]

Hyperpolarization is necessary in this case for faster deinactivation (returning to active state) of Na^+ channels since they inactivate (become inactive) during depolarization. Once Na^+ channels are deinactivated they can activate again if membrane potential exceeds the threshold.

Because Na^+ channels stay inactive for some time, it is not possible to excite a neuron. This period of time is defined as absolute refractory period. There is also a relative refractory period, followed by absolute refractory period, during which it is possible to excite the cell with stronger stimuli.

The overall result of this process, namely, rapid depolarizing Na^+current (Figure 2.1-5, I_{Na}) followed by delayed rectified repolarizing K^+ current (Figure 2.1-5, I_K) combined with inactivation of Na^+ voltage-gated channels, is a positive spike of membrane potential known as action potential (AP) [7].

Each individual ion channel opens in all-or-none fashion for variable duration of time that is unpredictable and determined by local random thermal and chemical activities on the nanoscale level. Thus, its behavior is stochastic (Figure 2.1-7).

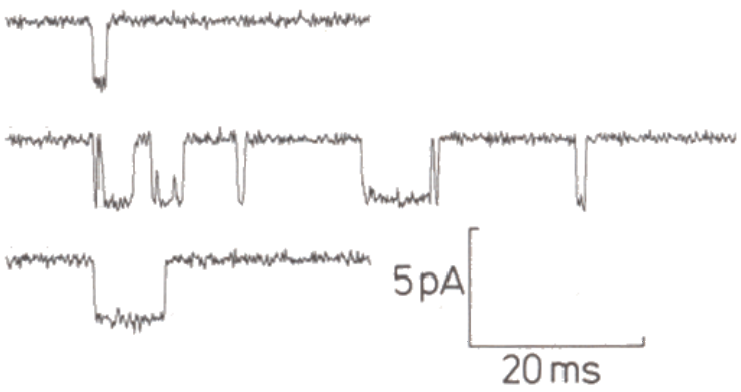


Figure 2.1-7 Single ion channel recording [8]

However, having a multitude of ion channels on a membrane surface averages single channel behavior. In fact, average open time is an intrinsic property of a given ion channel type. Thus, aggregated response of ion channels, expressed as increase in conductance, can be described without relation to the channels at all. Therefore, it is deterministic.

The variety of ion channels is not limited to voltage-gated Na^+ and K^+ ion channels. There are thousands of various ion channels that shape action potential. Having such a variety of ion channels nervous system achieves rich information processing capabilities with computation done by spike generation and synaptic transmission (discussed further) and communication done by spikes.

An example of another type of voltage-gated K^+ channels is M-type (muscarine-sensitive) K^+ channels, which contribute to the effect of spike frequency adaptation, SFA (an ability of a neuron to accommodate or adapt to increased stimuli and either stop or reduce firing) depicted in Figure 2.1-8

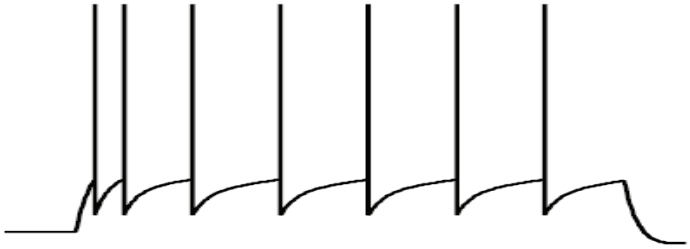


Figure 2.1-8 Membrane potential waveform with spike rate adaptation [9].

M-type K^+ channels require several hundred milliseconds to activate in response to depolarization. As a result, they provide an additional conductance path for K^+ ions, which leads to reduction of membrane input resistance, charge leak, and increase in time spent for accumulation enough charge on the membrane capacitor in order to reach the threshold and trigger the action potential.

Distribution of ion channels on a membrane surface creates a functional map of excitability of a neuron reflected in: enhanced propagation of action potential back to dendritic tree, propagation of dendritic impulse to the soma and axon hillock, regeneration of spike at the nodes of Ranvier, and many others (for morphology see Figure 2.1-1) [4]. Yet another degree of computational diversity of nervous system is the variety of neuron types, individual neurons, and their spiking patterns (Figure 2.1-9).

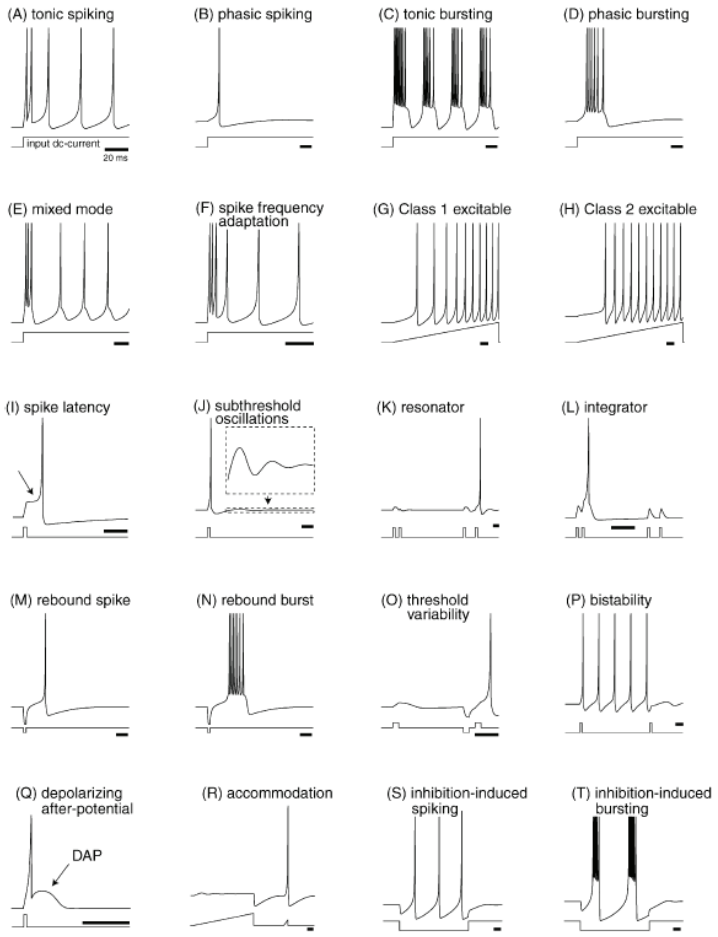


Figure 2.1-9 Summary of the neuro-computational properties of biological spiking neurons [10]

2.1.2 Synaptic transmission

In section 2.1.1 it is assumed that action potential is invoked by artificially injected current. In reality, a neuron receives the current from other neurons through synaptic transmission. A synapse is a connection between two neurons (Figure 2.1-1). There are electrical and chemical synapses. Electrical synapse provides bidirectional cytoplasmic continuity between two cells by means of gap-junctions (Figure 2.1-10). As a result, ion current can flow through low resistive path between connected cells. For fast depolarization of a post-synaptic cell its input resistance has to be larger compared to that of a pre-synaptic neuron. The latency of signal transmission between two cells is much

lower than that of chemical synapse. The primary function of electrical synapses is to provide firing synchrony between connected cells, especially between large groups of neurons performing the same task.

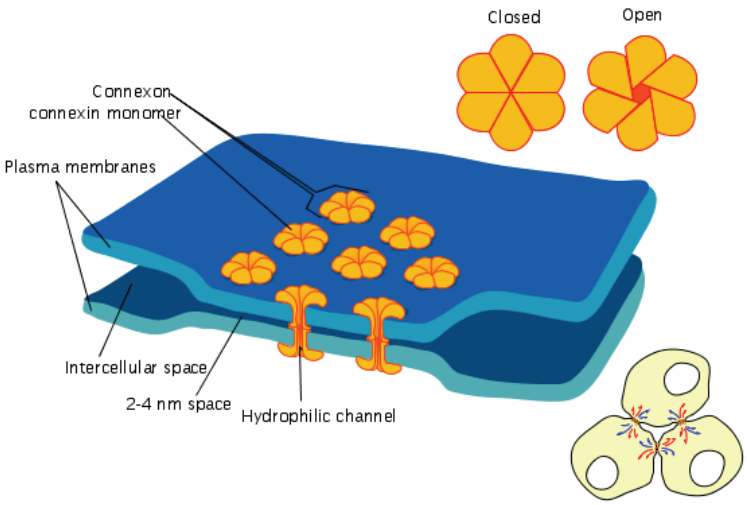


Figure 2.1-10 Gap junction

Properties of chemical synapses (Figure 2.1-11) are much more diverse, and their functionality is more complex. Action potential of a pre-synaptic cell, after it was generated at the axon hillock, is transmitted by the axon (see cell morphology in Figure 2.1-1) to axon terminal practically without a loss but with some delay. When it arrives at the terminal it causes voltage-gated Ca^{2+} channels to open and Ca^{2+} ions to enter the terminal (Figure 2.1-11, 1). This increase in Ca^{2+} ion concentration inside the terminal causes synaptic vesicles (small containers made of lipid bilayer) to fuse with the membrane at the place of active zone (part of the membrane looking at synaptic cleft) and release the neurotransmitter (a chemical that can bind to receptors) into synaptic cleft (small 20-40 nm gap between pre- and post-synaptic membrane) (Figure 2.1-11, 2-4). Neurotransmitter diffuses across the cleft, binds to the receptors, and causes them to activate and to open ion channels (Figure 2.1-11, 5), which results in rapid increase of membrane conductance in a post-synaptic cell.

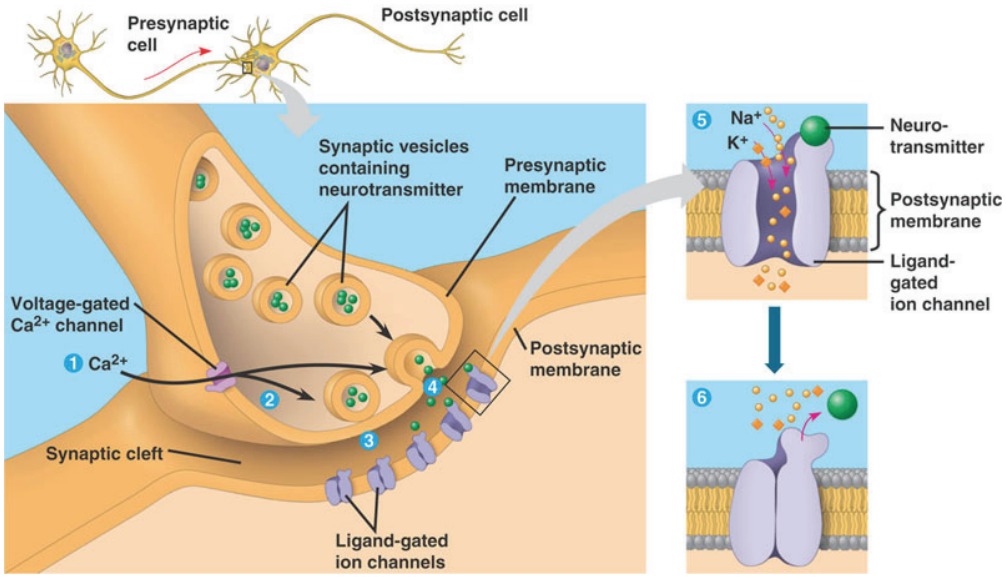


Figure 2.1-11 Chemical synapse

Receptor and its effector, an ion channel, can either compose the same unit, in which case this unit is a ligand-gated ion channel also known as ionotropic receptor, or they can be distinct units. In the latter case, the effector can be located anywhere on the membrane. It is activated via second messenger proteins synthesized and freely distributed across the cell because of receptor activation, and receptor is said to be metabotropic. After neurotransmitter is released, it diffuses out of the cleft and rapidly hydrolyses. Channels start to close in a random fashion (Figure 2.1-11, 6), which results in exponential decay of local membrane conductance.

Although process of chemical synapse activation is slower than simple signal transmission in the case of electrical synapse, it introduces several parameters into the system: 1) spread of neurotransmitter release, 2) concentration of neurotransmitter and its type, 3) receptor function and the way receptors activate ligand-gated ion channels, 4) presence and concentration of voltage-gated ion channels in post-synaptic neuron at the sight of the synaptic cleft (referred as synaptic density), 5) rate of neurotransmitter concentration reduction after its release in the cleft, 6) single ion channel transfer function, and others.

Spread of neurotransmitter after its release determines the impact and delay of signal transmission: it can be directed and fast as in case with synaptic cleft and active zone, or it can be more diffuse and slower playing a role of modulator.

Because each vesicle contains thousands of neurotransmitter molecules and there are only a few required for a single receptor to activate, relatively weak pre-synaptic spike can invoke a normal increase in post-synaptic conductance. Thus, the system provides a way for discrete signal propagation without signal loss.

The relation between receptor type and transmitter type is ONTO: several distinct receptor types (even with distinct functions) can utilize a single transmitter type. Receptor function can be either excitatory or inhibitory and its activation function depends on whether it is ionotropic (mostly responsible for reflexes) or metabotropic (mostly responsible for learning).

In addition to the ligand-gated channels, synaptic cleft may contain voltage-gated channels (Na^+ channels, for example) that enhance localized increase in membrane conductance, and contribute to local subthreshold potential spike and excitability of the cell.

The main differences between action potential and localized potential spike at the synaptic cleft are the following:

Voltage-gated Na^+ and K^+ channels open sequentially: first, Na^+channels convert potential chemical energy, stored outside of the membrane and determined by the concentration gradient of Na^+ ions, into kinetic energy of positive voltage spike, and second, K^+ channels do similar process to K^+ ions and their potential energy, which is converted to kinetic energy of negative voltage down stroke. Voltage-gating is reflected in regenerative feature for a multitude of ion channels. If one Na^+ channel opens, it depolarizes the membrane locally because of positive inward current that the channel permits to flow. This local depolarization results in surrounding channels opening in response to potential increase. Consequently, this action elevates local potential even more and propagates potential wave further laterally relative to the membrane. As a result, all channels open rapidly. K^+ current follows Na^+ current according to properties of K^+ channels. Overall, the result of this process is all-or-none action potential.

Ionotropic receptors open in response to neurotransmitter binding from extracellular side. The effect of this opening is limited since it is not regenerative and doesn't have all-or-none action. Channels open rapidly elevating membrane conductance and potential locally due to the abrupt increase in neurotransmitter concentration, but they close with exponential decay according to their stochastic properties (Figure 2.1-12, upper trace). At the same time, single-channel conductance stays constant and behaves as a switch.

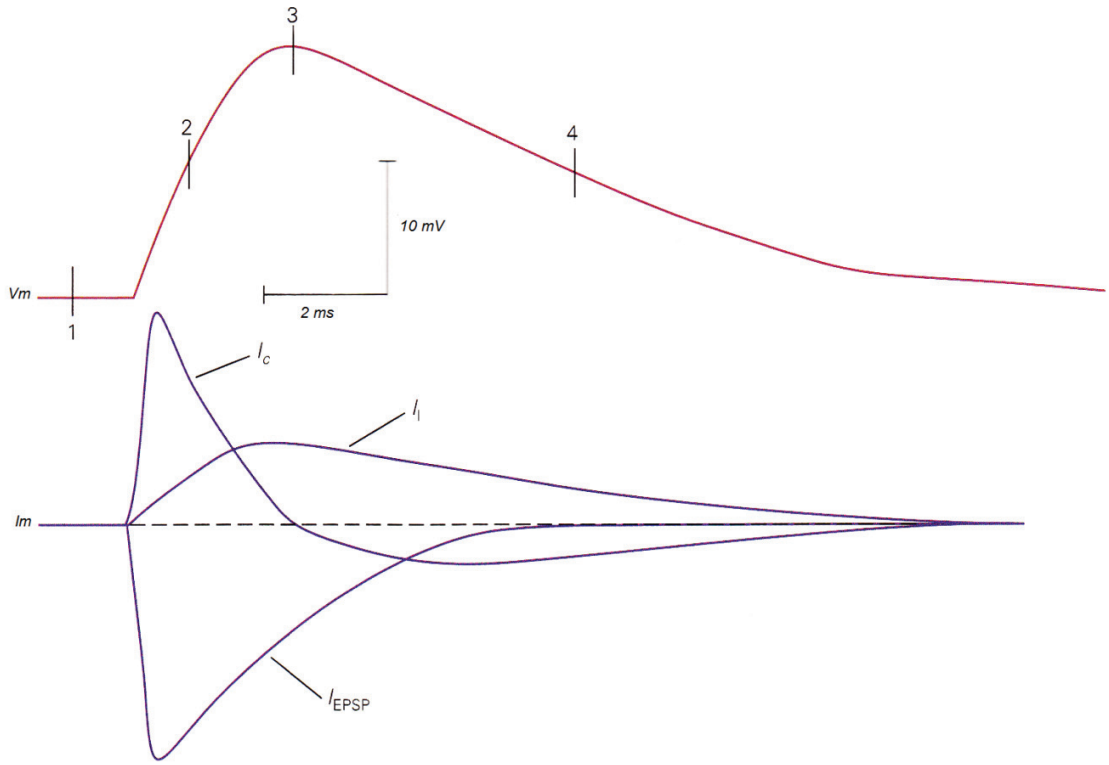


Figure 2.1-12 Membrane potential and current traces during process of excitatory synaptic transmission [4]

The peak of this local spike in the membrane conductance (known as synaptic weight) is determined by neurotransmitter and receptor concentration. The resulting potential wave is called excitatory post synaptic potential (EPSP) in case of excitatory receptors or inhibitor post synaptic potential (IPSP) in case of inhibitory receptors. Similar to the case of resting potential, reversal (equilibrium) potential of a particular ion

in the cleft is indicative of concentration gradient force on this ion and is the same across the membrane. However, since receptors may be permeable to several ion species each at specific conductance value (for example in neuromuscular junction receptors conduct both K^+ and Na^+ ions), the reversal potential for all ion species in the cleft is most likely to be different from that of resting potential (reversal potential for the membrane at rest). Thus, similar to equation (2.1-1) the reversal potential of receptors in the cleft can be found as a weighted average of equilibrium potentials of ions that these receptors are permeable to, where the weights are specific ion conductances, intrinsic properties of the receptor type. In the case of neuromuscular junction the reversal potential is zero.

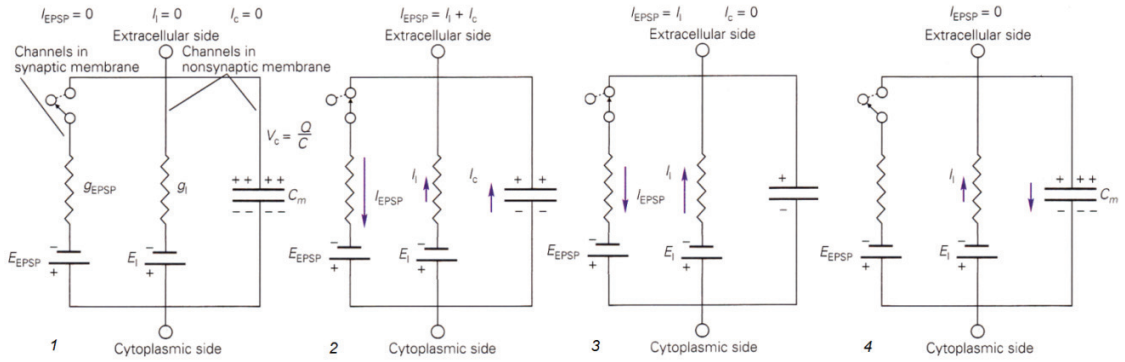


Figure 2.1-13 Process of excitatory synaptic transmission: electrical aspect [4]

The process of excitatory synaptic transmission from the perspective of post-synaptic cell can be described using equivalent electrical circuit (Figure 2.1-13) and corresponding waveform (Figure 2.1-12) as a sequence of several steps: 1) At steady state post-synaptic potential is equal to resting potential. Synaptic equivalent conductance g_{EPSP} that varies according to stochastic properties of channels is zero. Equivalent reversal potential E_{EPSP} stays constant throughout the process; 2) g_{EPSP} increases rapidly since the majority of channels open. Synaptic current I_{EPSP} quickly reaches its peak starting from zero and contributing primarily to capacitive current I_C, thus, depositing positive charge on the membrane capacitor and elevating the membrane potential; 3) I_{EPSP} and I_C peak and drop. Leakage current I_l due to resting channels surrounding the cleft takes over. When $I_{EPSP} = I_l$, I_C becomes zero and membrane potential reaches its

peak. 4) I_C changes its direction returning the acquired charge to extracellular side through resting channels. V_m slowly decays to its resting value.

Due to the fact that leakage current takes over only in step (3) ratio of g_{EPSP} value at its peak to g_l (the equivalent resting conductance) value defines the magnitude of potential peak: the larger this gap the more charge is deposited during step (2), and the larger EPSP peak.

Among most common excitatory ionotropic receptors are L-glutamate activated receptors: NMDA receptor (N-methyl-D-aspartate) permeable to Ca^{2+}, Na^{+}, K^{+}; and non-NMDA receptors (permeable to Na^{+} and K^{+}): AMPA receptor (α-amino-3-hydroxy-5methyliosoxazole-4-propionic acid) and kainite receptor.

Among most common inhibitory receptors are GABA activated ionotropic GABA $_A$ Cl^{-} channels and metabotropic GABA $_B$ receptors that activate K^{+} channels.

The sequence of steps describing excitatory synaptic transmission is applicable to inhibitory synaptic transmission. However, in this case ion channel species are Cl^{-} (for GABA $_A$), and I_{IPSP} is an outward current. Indeed, according to Table 2.1-1, chemical concentration gradient force affects Cl^{-} ions and drives them inside the cell resulting in negative charge influx as a response to increased g_{IPSP}, which corresponds to the outward positive current. However, reversal potential of the channels E_{IPSP} corresponds to equilibrium potential of Cl^{-} ions since channels permit only this type of ions. This potential is the same (or slightly less) as membrane potential at rest, which results in zero or small Cl^{-} current through the channels. How does opening of Cl^{-} channels in inhibitory synapse with no current can affect membrane potential? The explanation of this process is described in the next section.

2.1.3 Signal integration and modulation

A variety of synaptic transmission events received in time by a single neuron, their polarity, strength, and other parameters described in this text define response of this neuron through the process known as neuronal integration.

The primary function of neuronal integration is summing ion currents from synapses of opposite polarity. As it was shown in section 2.1.2, E_{IPSP} of ionotropic GABAergic synapses is near resting membrane potential, and g_{IPSP} increase produces none or small outward I_{IPSP}. However, in the presence of simultaneous I_{EPSP}, the effect of increased g_{IPSP} is prominent, because in this case increased overall membrane conductance prevents some part of I_{EPSP} from depositing on membrane capacitance as a charge and instead leaks away from the cell as I_{IPSP}, although using different ion species. Thus, inhibitory synapse provides a conductive shunting path for any excitatory current in the cell. This process is called shunting inhibition (Figure 2.1-14). Shunting inhibition patterns temporal activity of spiking neurons (sculpturing).

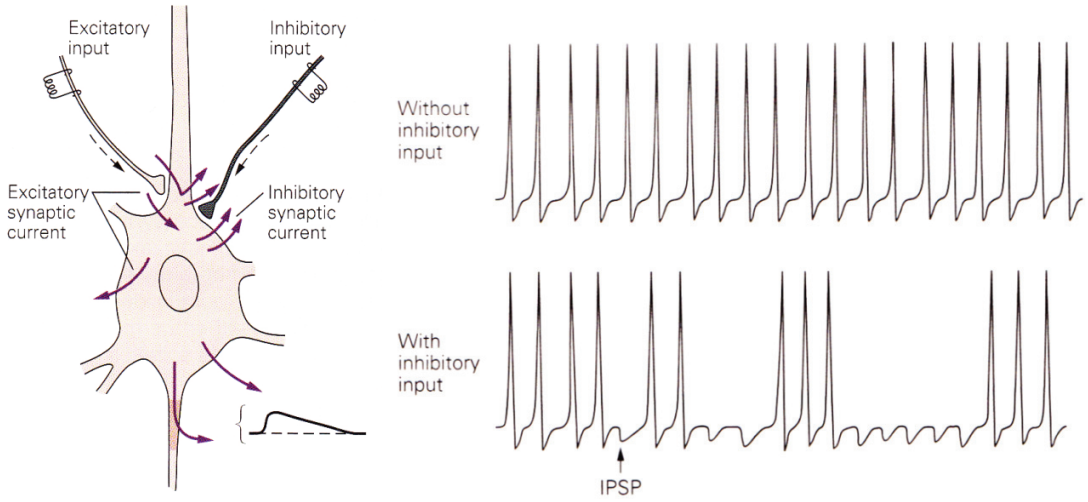


Figure 2.1-14 Shunting inhibition and its sculpturing effect on neural signal [4]

Because reversal potential of Cl^- receptor is very close to resting potential, it is relatively easy for Cl^- ions to accumulate inside of a cell with successive inhibitory events. This, however, may reverse the action of synapse from inhibitory to excitatory if accumulation achieves the point when concentration gradient of Cl^- reverses the direction of chemical force.

One of the most important properties of neuronal integration is spatiotemporal summation of signal (Figure 2.1-15).

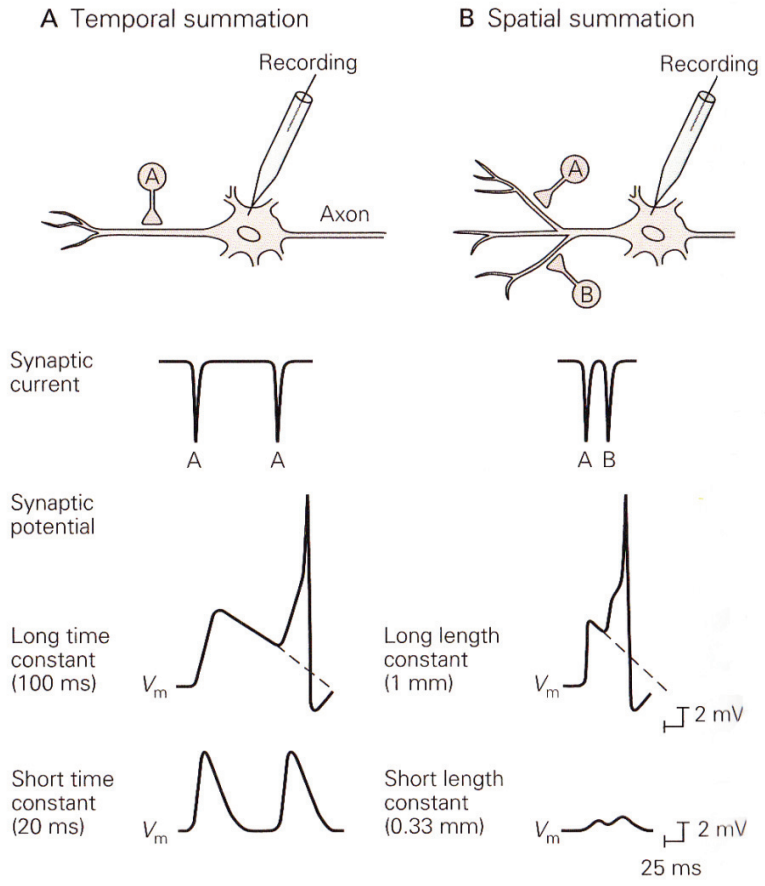


Figure 2.1-15 Spatiotemporal neuronal integration [4]

Larger membrane capacitance holds more charge and requires more time for charging and discharging, which results in longer time constant. Capacitive current is usually the fastest (Figure 2.1-12), and leakage conductance is smaller than synaptic conductance. Therefore, it is primarily $\frac{C_m}{g_{EPSP}}$ that determines the time of depolarization in case of EPSP, and $\frac{C_m}{g_{leak}}$ determines time of repolarization. As a consequence, the membrane with larger C_m repolarizes slower and is more likely to generate action potential as a result of consecutive EPSPs.

Another form of integration is spatial summation. Each dendrite can be represented as a network of compartments related to segments of a cell with

approximately constant lateral (series) resistance and associated transverse (shunt) membrane RC sub-circuit (Figure 2.1-16).

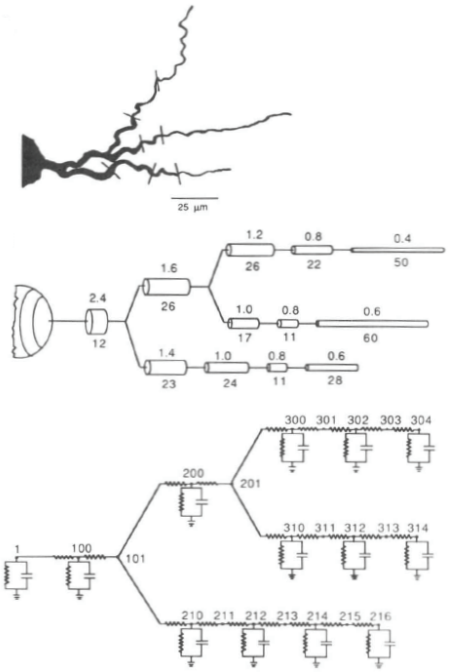


Figure 2.1-16 Compartmental division of dendritic tree [11]

Thus, a passive component of dendritic tree can be viewed as a network of non-inductive leaky transmission lines. If length constant of this line is such that its shunting properties are weak relative to conductive properties of lateral component (smaller transverse capacitance and conductance, but larger lateral conductance), then current loss from the source (synapse) to axon hillock is minimal, and length constant is said to be large. With larger length constant more current is delivered to axon hillock, more chances are that membrane produces a spike. The effect is opposite with smaller length constant.

Interestingly, the type of synaptic connection is tied to the function of synapse (Figure 2.1-17).

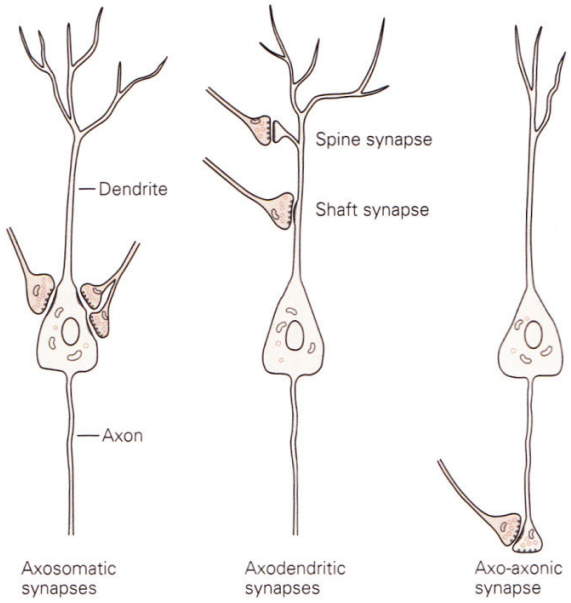


Figure 2.1-17 Types of synaptic connections [4]

Axosomatic synapses are usually inhibitory: they provide shunting path for currents flowing from dendritic tree to axon hillock. Axondendritic synapses provide excitatory integrative function. Axo-axonic synapses are usually modulatory: they control neurotransmitter release. The role of modulatory function in synaptic integration is explained further.

Signal modulation is achieved by a variety of means. Simple signal modulation resulting from activation of voltage-gated M-type K^+ channels and consequent effect of SFA has been shown in section 2.1.1.

Another example of signal modulation is activation of NMDA ionotropic receptor (Figure 2.1-18) in response to the presence of extracellular glycine as a cofactor, glutamate as a major transmitter, and depolarization as a voltage condition. The channel is unique in the sense that it is gated by two conditions: chemical and electrical. Besides, electrical condition is activated in unique manner as well: under slight depolarization blocking Mg^{2+} ion is expelled from the channel by electrostatic repulsion.

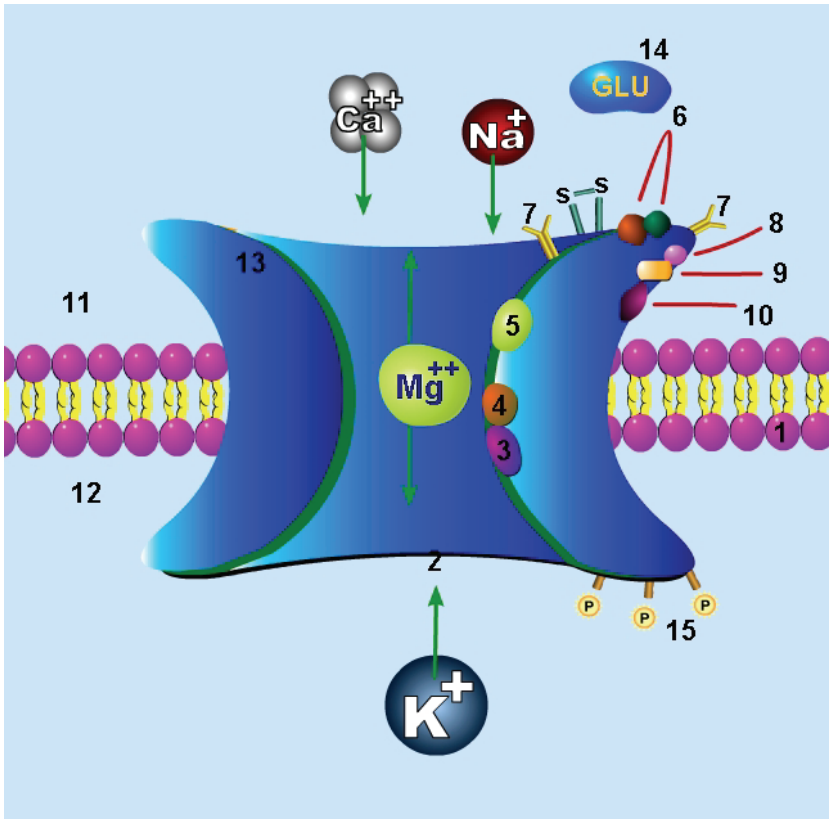


Figure 2.1-18 NMDA receptor. Legend: 1. Cell membrane, 2. Channel blocked by Mg2+ at the block site (3), 3. Block site by Mg2+, 4. Hallucinogen compounds binding site, 5. Binding site for Zn2+, 6. Binding site for agonists(glutamate) and/or antagonist ligands(APV), 7. Glycosilation sites, 8. Proton biding sites, 9. Glycine binding sites, 10. Polyamines binding site, 11. Extracellular space, 12. Intracellular space

The channel is permeable to Ca^{2+}, Na^{+}, K^{+}. Activation of the channel is achieved if the cell has already been depolarized locally (several pre-synaptic spike trains arrived and, as a result, EPSPs have been generated due to non-NMDA receptors) and therefore electrical condition is met. Another way to meet this condition is to generate action potentials in the cell by any means, which can back-propagate to the synapse of interest. Only if electrical condition is met and blocking Mg^{2+} ion is out of the channel, receptor response to the transmitter reveals itself. The response is slower than that of non-NMDA ionotropic glutamate receptors. As a result, contribution of the channel is primarily in increasing potential during late transient of EPSP allowing more inward current (Figure 2.1-19). This current is mostly due to Ca^{2+} ions.

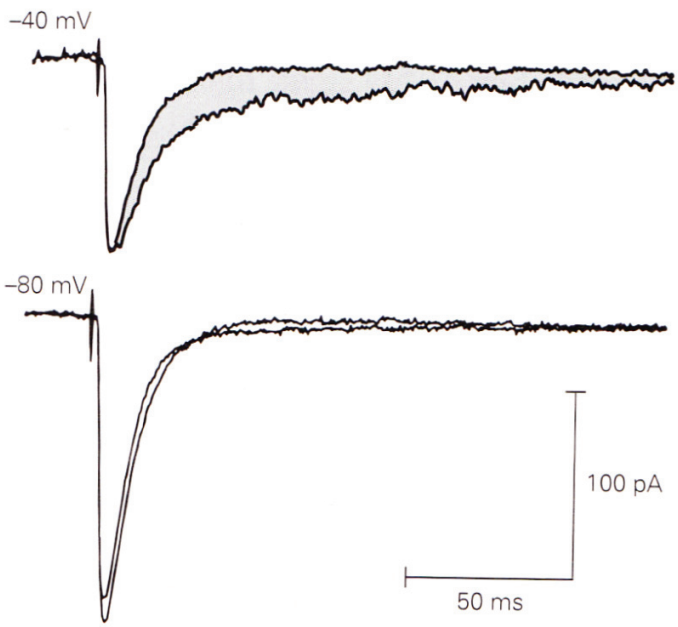


Figure 2.1-19 Synaptic current trace at -40 mV and -80 mV. Synapse with blocked NMDA is compared to synapse with functioning NMDA. Shaded area is the difference [4]

However, the role of NMDA is not limited to the increase in inward EPSP current and consequently, the excitability of the cell. The fact of electrical condition in NMDA receptors introduces casualty into pre- and post-synaptic activity if the effect of back-propagating action potentials is considered. As a result, NMDA receptor plays role of a spike coincidence detector and alerts by elevating local Ca^{2+} ion concentration if casualty is detected [12]. Consequently, stable presence of Ca^{2+} ions in high concentration in post synaptic elongation of the dendrite, called spine (see Figure 2.1-17 center image), may strengthen synaptic connection via downstream second messengers generated in the similar way as in the case of metabotropic receptors (discussed in 2.1.2). Such increase in strength is a result of amplified excitability of ionotropic AMPA receptors as a response to second messengers in short term. However, if spike detection is steadily repeated over a long period of time, it invokes conformational changes in the spine expressed as increase in the number of AMPA receptors and even more significant

morphological changes, such as enlargement of related dendrite, creation of new spines, and others [13]. Overall this process of the increase in synaptic strength is called long term potentiation (LTP), which is a form of synaptic plasticity (an ability of synapses to change its strength in response to spikes). The reverse process, weakening of synaptic strength, is called long term depression (LTD).

Notwithstanding the fact that LTP and LTD have opposite effects on a synapse, they are closely related, since both of these processes originate from NMDA receptor properties. General rule for excitation is the following: if pre-synaptic EPSP arrives a few milliseconds before post-synaptic cell generates AP, then synapse exhibits LTP; however, if pre-synaptic EPSP arrives few milliseconds after post-synaptic cell generates AP, then the synapse exhibits LTD. This type of synaptic plasticity is called spike time dependent plasticity (STDP). It is one of the most controversial and actively researched types of plasticity. The controversy comes from various aspects: for some neurons the rule is inverted; there is a large variability in timing window between LTD and LTP among neurons and synapses of the same neuron, and others [12]. Besides, STDP is affected by various processes: spiking frequency, threshold of depolarization, distance of synapse from axon hillock, reinforcement of back-propagating AP strength, AP width [14], and others.

At the same time, the general accepted cause of LTD and LTP induction in STDP is difference in Ca^{2+} current transients: brief and strong postsynaptic Ca^{2+} elevations signal LTP; smaller, more prolonged Ca^{2+} transients induce LTD. Simplified explanation of this can be done on the basis of Ca^{2+} dynamics. Voltage-dependant Ca^{2+} channels (VDCCs) located on a membrane on the branches of a dendritic tree generate background Ca^{2+} concentration in the cell in response to AP [14]. If EPSP precedes AP, the background Ca^{2+} is at its lowest peak, and Ca^{2+} concentration gradient across the membrane is the largest. As a consequence, Ca^{2+} influx is the strongest. However, if EPSP follows AP, VDCCs are activated on a way of AP propagating back to the synapse spine, where EPSP is about to reveal itself. Elevation of Ca^{2+} concentration in the cell due to Ca^{2+} flux from VDCCs occurs before NMDA opening. As a result, NMDA exhibits smaller and more gradual Ca^{2+} influx.

It is imperative to understand that above explanation is rather hypothetic, because STDP is the area of active ongoing research, and new, unknown before, contradicting results actively contribute to the overall picture of STDP as a much more complex process.

On a long term due to the response properties of NMDA receptors LTP is spike frequency-dependent: high frequency stimulation leads to LTP, whereas low frequency stimulation leads to LTD. Brief high-frequency stimulation results in strong postsynaptic depolarization and NMDA receptor activation, whereas sustained low-frequency stimulation evokes less NMDA receptor-dependent Ca^{2+} influx leaving the rest for VDCCs.

Another interesting property of NMDA receptor is dynamic reduction of glutamate affinity as a response to glutamate exposure in the presence of Mg^{2+} blocker in NMDA channel. As a consequence, it establishes the size of a time window for efficient LTP: depolarization immediately after glutamate release opens channel more efficiently than later in time [15].

NMDA is not the only mediator of LTP and LTD, neither its dynamics is precisely described in this text, since process of STDP is not completely understood.

One of the most important contributions of STDP is classical conditioning. Classical conditioning is based on associative learning. It involves two types of stimuli/response [16]: 1) Unconditioned stimuli invoke innate, often reflexive response, such as salivation when sensing food. 2) Conditioned stimuli are neutral, for example, hearing a ringing bell. After both food and ringing bell are presented, for example, to a dog several times, eventually the dog learns to salivate in the presence of ringing bell without presence of food.

At synaptic level classical conditioning in its simplistic case is based on increasing strength of synaptic transmission for a synapse associated with conditioned stimuli while generating APs by means of EPSPs from the synapses associated with unconditioned stimuli. As a result, NMDA role as a coincident detector expresses itself in LTP. Whether classical conditioning is based on STDP or frequency dependent plasticity is defined by signaling pattern and type of stimuli complexity.

Compared to ionotropic receptors, in which receptor and effector constitute the same unit, metabotropic receptors activate/deactivate their effectors via chain of biochemical processes transpired in cytoplasm. The processes employ second messengers. As a result, activation takes longer and its duration lasts longer as well. Besides, the range of activation is global to the cytoplasm due to freely distributed second messengers.

Another property of metabotropic receptors is the duality of their action: under different conditions they can decrease or increase channel opening. The main function of metabotropic receptors, however, is modulatory synaptic actions, such as changes in resting potential, input resistance of the cell, time constants, threshold potential, action potential duration, firing patterns, and others. These actions can be typified by effector channel targets and the corresponding effects they produce: 1) Channels at pre-synaptic axon terminals (transmitter release); 2) Ionotropic receptors (synaptic potential); 3) Resting and voltage-gated channels (excitability and firing).

An example of type (3) modulation is disabling or decreasing effect of spike frequency adaptation by means of metabotropic muscarinic acetylcholine- (ACh) activated receptors. These receptors are present along with ionotropic nicotinic receptors in excitatory synapses. Upon synaptic transmission they signal via second messengers to M-type K^+ channels and as a consequence, disable their activation. The result of this is increase in input membrane resistance, increase in current required for cell depolarization, increase in cell excitability, and reduction of spike rate adaptation effect.

Another example is modulation of NMDA mediated STDP by D1-like dopamine receptors. Activated by dopamine, these receptors enhance sensitivity of LTP induction by expanding the timing window and lowering the number of repetitive pairings required for effective induction. Besides, activation of dopamine receptors converts LTD into LTP in hippocampal synapses [17]. These recent findings reveal dopamine as an enhancing modulator for learning in synapses where D1-like dopamine receptors are expressed.

2.2. Neuron models

In order to reverse-engineer biophysical processes described in section 2.1 using different implementation technology for the purpose of their application in various fields, it is essential to understand which processes nature tries to mitigate and how and which ones are essential mechanisms that contribute to the variety of system dynamics. In this sense it might be possible to avoid effects mitigated by the nature if implementation technology allows that. However, caution has to be taken, because not all processes are fully understood and the ones that seem unwanted now may become having considerable contribution in the future.

The world of neuronal models is much more diverse compared to what is presented in this text; however it is very far from being diverse if compared to the variety of biochemical processes, small part of which is presented in section 2.1. There are models that consider each individual ion channels [18], [19] and model membrane surface dynamics as accurate as possible. There are stochastic spiking neuron models [20]. There are compartmental models that consider impact of dendritic tree on internal current dynamics of the soma by slicing a cell into compartments [11]. There are single-compartment models that assume all synapses terminate on soma and impact axon hillock directly. It is the application targeted by the model and its requirements that determine the model choice.

In this section three most commonly used single-compartment models of membrane dynamics are reviewed: integrated-and-fire model (IF), which is the oldest and the least accurate, but fast to compute; Hodgkin-Huxley model (HH), which is considered as the most accurate, but most difficult to solve and compute; Izhikevich model (IZ), a phenomenological model that is claimed to be as accurate as Hodgkin-Huxley model, but not as computationally prohibitive.

2.2.1 Hodgkin-Huxley model

HH model [7] is considered the most biologically plausible and intuitive (its variable set has biological interpretation). Developed by pioneers, who discovered mechanics behind action potential, the model is still widely used today if accuracy of neuron simulation is of the first priority. The equations of the model are the following:

$$
\begin{aligned}
&C_m \frac{dV}{dt} = I_{inj} - I_{leak} - I_{Na} - I_K, \\
&I_{leak} = \overline{g_{leak}}(V - E_{leak}),\ I_{Na} = \overline{g_{Na}} m^3 h (V - E_{Na}),\ I_K = \overline{g_K} n^4 (V - E_K) \\
&\tau_i(V) \frac{di}{dt} = i_\infty(V) - i,\ \ \tau_i(V) = \frac{1}{\alpha_i(V) + \beta_i(V)},\ \ i_\infty(V) = \frac{\alpha_i(V)}{\alpha_i(V) + \beta_i(V)} \\
&i = \{m, h, n\} \\
&\alpha_n(V) = \frac{A_{\alpha n}(V + B_{\alpha n})}{exp\left(\frac{V + B_{\alpha n}}{C_{\alpha n}} - 1\right)},\ \ \beta_n(V) = A_{\beta n} exp\left(\frac{V}{B_{\beta n}}\right) \\
&\alpha_m(V) = \frac{A_{\alpha m}(V + B_{\alpha m})}{exp\left(\frac{V + B_{\alpha m}}{C_{\alpha m}} - 1\right)},\ \ \beta_m(V) = A_{\beta m} exp\left(\frac{V}{B_{\beta m}}\right) \\
&\alpha_h(V) = A_{\beta h} exp\left(\frac{V}{B_{\beta h}}\right),\ \ \beta_h(V) = \frac{1}{exp\left(\frac{V + B_{\alpha h}}{C_{\alpha h}} + 1\right)}
\end{aligned}
\tag{2.2-1}
$$

In this equation set C_m is membrane capacitance, V is membrane potential. A set of currents I_{inj}, I_{Na}, I_K, I_{leak} is composed of injected current, Na^+ and K^+ currents, leakage current due to Cl^- and other than Na^+ and K^+ ions. A set of constant conductances $\overline{g_{leak}}$, $\overline{g_{Na}}$, $\overline{g_K}$ is represented by leakage conductance, and maximum Na^+ and K^+ conductances during action potential. A set of reversal potentials, E_{leak}, E_{Na}, E_K, comprises leakage Nernst potential, Na^+, and K^+ reversal potentials. Gating variables, m, h, n for transient conductance modeling, correspond to Na^+ activation and inactivation variable, K^+ activation variable respectively. Each gating variable i models a gate of ion channel and is responsible for activation or inactivation of related current during action potential with effect of exponential decay towards voltage-dependent value $i_\infty(V)$ with voltage-dependent rate $\tau_i(V)$. $\alpha_i(V)$ and $\beta_i(V)$ are gate opening and closing rates respectively (in case of h it is opposite since h models inactivation gate). Constants

$A_{subscript}$, $B_{subscript}$, $C_{subscript}$ are obtained by experimental fitting during matching to the desired curve. Typical result of fitting produces $i_\infty(V)$ and $\tau_i(V)$ such as in Figure 2.2-1.

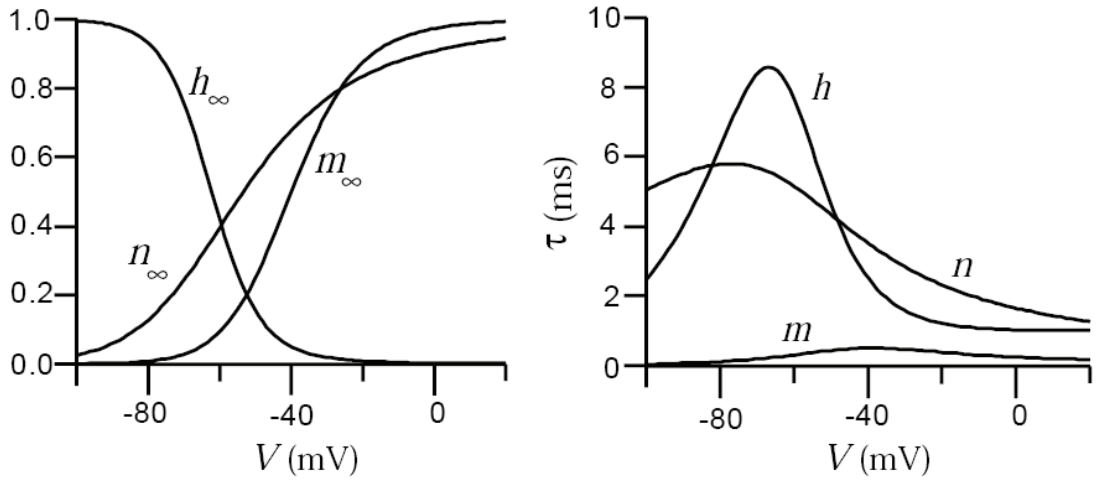


Figure 2.2-1 $i_\infty(V)$ and $\tau_i(V)$, $i = \{m, h, n\}$ [9]

In Figure 2.2-1 time constant $\tau_h(V)$ of gating variable h has evident bump about $V = -70\,mV$. As it is seen from the equation, variable h due to its negative feedback models inactivation process (halting I_{Na} flow, discussed in 2.1.1). This corresponds to engaging inactivating gate in voltage-gated Na^+ channels. However, in order to repel this gate from the channel (deinactivate) faster, it is necessary to hyperpolarize the cell. Thus, this bump corresponds to faster recovery of h within this potential vicinity – the process of deinactivation.

The model variables can be used either as current densities or as absolute values. In the former case, variables $\overline{g_{type}}$, C_m, I_{type} are expressed as a value per unit area of the membrane. Resting potential E_{rest} is not explicitly defined, but it is assumed that $V = E_{rest}$ is at the steady state.

Operation of this model is as following. A variety of currents are integrated on membrane capacitance C_m as a charge. At rest and with no disturbances $V = E_{rest}$, and the net current across membrane is zero. If the system is disturbed with inward

depolarizing current I_{inj}, the resulted increase in membrane potential V increases $m_\infty(V)$ and decreases $h_\infty(V)$. At some implicit threshold potential value both $m_\infty(V)$ and $h_\infty(V)$ are above zero and such that gating variables m and h lagging behind these values with rates $\tau_m(V)$ and $\tau_h(V)$ slowly open I_{Na}. This, in turn, results in larger increase of V, pushing $m_\infty(V)$ to 1 and $h_\infty(V)$ to 0. However, the rate with which these variables change are different, namely, $\tau_h(V)$ is more than $\tau_m(V)$ (Figure 2.2-1) and therefore, h approaches $h_\infty(V)$ slower than m does $m_\infty(V)$ (Figure 2.2-2).

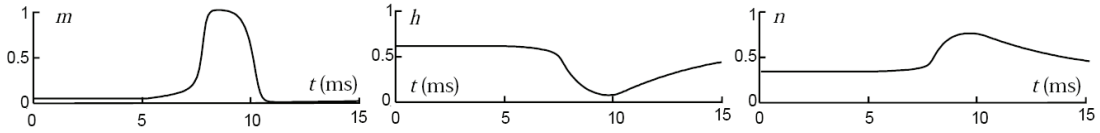


Figure 2.2-2 Dynamics of gating variables in HH model [9]

Thus, the effect of negative feedback from h lags behind the effect of positive feedback from m resulting in abrupt influx of I_{Na} (Figure 2.2-3).

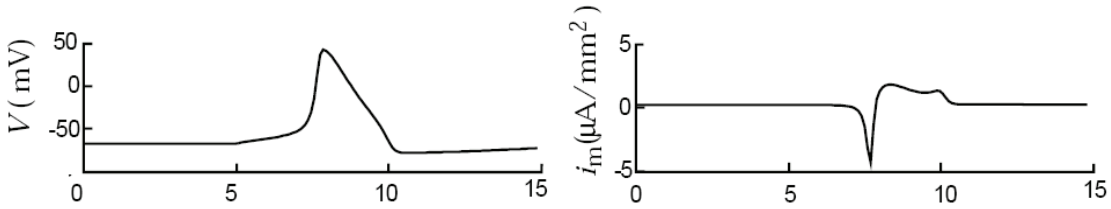


Figure 2.2-3 Membrane potential and current dynamics in HH mode [9]

However, this influx returns to zero leaving membrane potential to repolarize by itself. In order to facilitate faster repolarization, gating variable n provides positive feedback to the membrane equation. Since K^+ reversal potential is negative compared to that of Na^+, it results in negative outward current in the membrane equation repolarizing the membrane to the potential within the vicinity of resting potential or slightly below in order to aid deinactivaiton of Na^+ channels.

2.2.2 Integrate-and-fire models

This set of models is based on modeling of subthreshold membrane potential dynamics excluding explicit mathematical description of membrane voltage-gated K^+ and Na^+ conductances [21]. As a result, in its simplistic form, called leaky or passive integrate-and-fire, the model can be described as a parallel resistive-capacitive network (Figure 2.2-4, soma part).

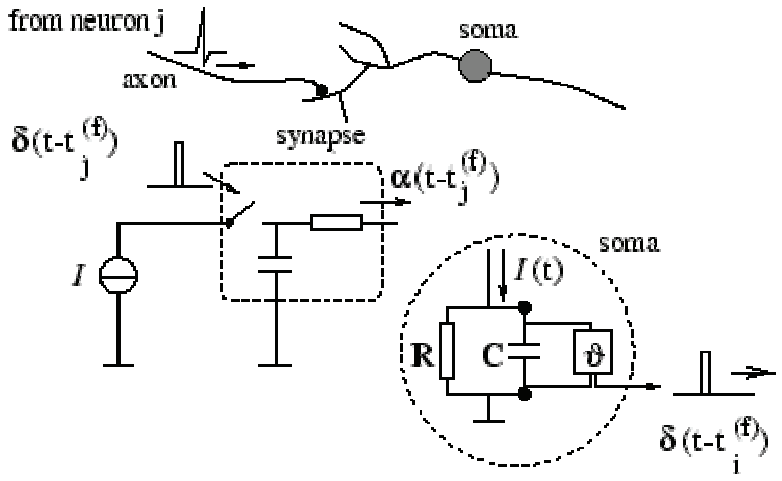


Figure 2.2-4 Circuit of Integrate-and-Fire model [22].

Mathematically, a simple passive IF model is described by the following set of equations:

$$C\frac{dV}{dt} = I_{inj} - I_{leak},\quad I_{leak} = \overline{g_{leak}}(V - V_{resting}),$$
$$if\ V \geq V_{threshold}: V = V_{reset}\ after\ t_{firing} \tag{2.2-2}$$

In these equations: I_{inj} and I_{leak} are injected and leakage currents of the cell respectfully, C and V are membrane capacitance and voltage respectfully, $V_{resting}$ is

resting potential of the cell, $V_{threshold}$ and t_{firing} are firing threshold potential and firing time respectfully, V_{reset} is a reset potential after firing, $V_{reset} < V_{threshold}$.

With added spike frequency adaptation discussed in 2.1.1 model equations become:

$$C\frac{dV}{dt} = I_{inj} - I_{leak} - I_{adap},\ \ I_{leak} = \overline{g_{leak}}(V - V_{resting}),\ I_{adap} = g_K(V - V_K),\ \tau_K\frac{dg_K}{dt} = -g_K$$
$$if\ V \geq V_{threshold}\text{: 1) } g_K = g_K + \Delta g_K,\quad 2) V = V_{reset} \text{ after } t_{firing} \tag{2.2-3}$$

In these equations newly added variables are: I_{adap} is adaptation current that models K^+ dynamics and spike rate adaptation, g_K is K^+ conductance, τ_K is time constant that defines g_K dynamics over time.

The model operates in the following way: certain part of the current I_{inj} is integrated on a membrane capacitor C as a charge forcing membrane potential $V$to rise. Another part of this current simply leaks through linear resistor $1/\overline{g_{leak}}$ as I_{leak}. Sometime after the membrane potential V elevates beyond the threshold potential $V_{threshold}$ (this time is defined by the firing time t_{firing}) V is reset to V_{reset} and the process repeats. However, there is another part of the membrane current, I_{adap}, which is subtracted from I_{inj} and leaks through non-linear conductance g_K. The value of g_K is defined by the firing rate since g_K is incremented by some value Δg_K after each spike. At the same time, g_K decays exponentially during the time between the spikes. As a consequence, mean value of Δg_K and therefore, mean value of I_{adap} is well defined after several spikes assuming that I_{inj} is at the steady state. If I_{inj} changes then g_K adapts to its new mean value after several spikes (Figure 2.1-8).

Model can be augmented with a refractory period in several different ways: 1) Interrupting dynamics of the model for some period of time after the reset: $V \rightarrow V_{reset}$; 2) Incrementing $V_{threshold}$ after each threshold with following exponential decay; 3) Adding a conductance similar to the one that models spike rate adaptation, g_K.

One of the major benefits of this model is its simplicity and small execution time if used in neural networks based on numerical integration methods. However, biological plausibility of this model in respect to the gamut of spiking patterns is limited.

2.2.3 Izhikevich model

IZ model is relatively new [3], however, it's been acquiring active use in neuroscientific computational systems. It is derived according to bifurcation theory and normal form reduction [23]. The goal of this model is to achieve biological plausibility comparable to that of HH model, but at the same time be lightweight computationally. The author claims that the model provides biological functions comparable to those of HH model. However, the model variable set doesn't have biological interpretation. Thus, the model is said to be phenomenological. The model equations are the following [24]:

$$C\frac{dv}{dt} = k(v - v_{rest})(v - v_{thresh}) - u + I, \qquad \frac{du}{dt} = a(b(v - v_{rest}) - u)$$
$$if\ v \geq v_{peak}: \begin{cases} v = c \\ u = u + d \end{cases} \tag{2.2-4}$$

In these equations: C is membrane capacitance, v is membrane potential, v_{rest} is resting membrane potential, v_{thresh} is threshold potential, v_{peak} is action potential escape limiting value, u is recovery variable that models Na^+ and K^+ currents, I is injected current, a is a parameter that describes the time scale of recovery variable u, b is a parameter that describes sensitivity of u to the subthreshold fluctuations of membrane potential, c is a parameter that describes after-spike reset value due to hyperpolarizing outward K^+ current and is typically set to a value less than v_{rest}, d is a parameter that describes after-spike reset of u, k describes sensitivity of v to the fluctuations of itself.

The model functions in the following way: injected current I is integrated as a charge on membrane capacitor C, which results in membrane potential fluctuations v. If v crosses v_{thresh}, the quadratic part of the capacitor equation, $k(v - v_{rest})(v - v_{thresh})$,

accelerates membrane potential dynamics, which results in a spike. At the same time, this high spiking value of v accelerates u, which provides negative feedback to v and to itself playing a role of K^+ ionic current at that time. Compared to v, which provides quadratic dependence to its acceleration, the equation that governs u has only 1st power dependence on v. Thus, u shaped by a and b, follows v. As a consequence, v has to be artificially reset in order to keep it in the plausible range. Thus, once v reaches v_{peak}, it is reset to value c. At the same time, u is incremented rather than reset, and thus, it "memorizes" previous spike dynamics and affects the refractory period. Variable u without term $b(v - v_{rest})$ would play a similar role as g_K plays in IF mode. However, this term gives it the functionality of controllable response to membrane potential dynamics, and therefore, enriches the model with types of dynamics such as intrinsically bursting, chattering, fast spiking, low-threshold spiking, thalamo-cortical spiking, resonating and other dynamics [3].

A chart describing biological plausibility and computational complexity of models assembled by the author of IZ model is presented in Figure 2.1-1.

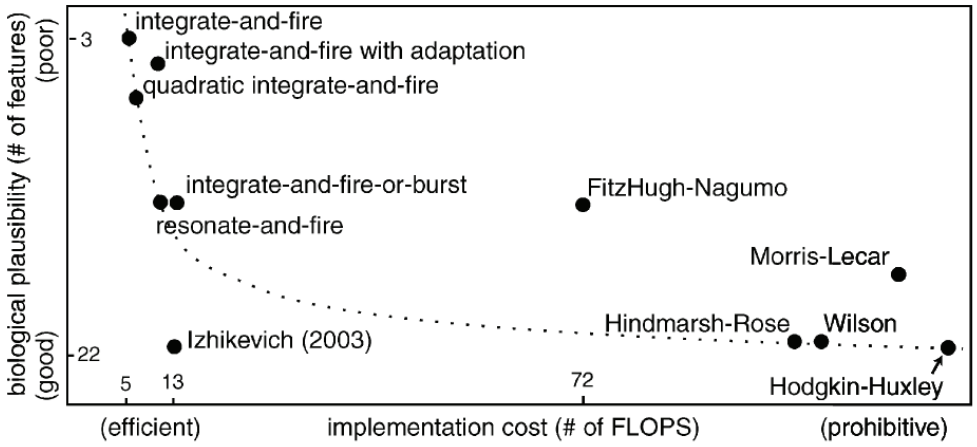


Figure 2.2-5 Comparison of biological plausibility and implementation cost of neuron models [10]

2.2.4 Simple post-synaptic conductance model

In its simplistic way the response of post-synaptic conductance to an AP arrived at the axon terminal can be modeled as a step with an exponential decay. Although this

type of conductance does not provide complex NMDA response, neither it counts for the synaptic delay (synaptic delay can be merged with axonal transmission delay) nor for finite slope of conductance rise (see section 2.1.2), it provides a simple way to interact with a membrane equation of any membrane model described above. This interaction is done by adding another current term $-I_{synapse}$ to the capacitor equation of corresponding membrane model [25]:

$$\begin{aligned} I_{synapse} &= -\eta\left(v - E_\eta\right) - \gamma\left(v - E_\gamma\right) \\ \frac{d\eta}{dt} &= -\lambda_\eta \eta \\ \frac{d\gamma}{dt} &= -\lambda_\gamma \gamma \end{aligned} \tag{2.2-5}$$

where v - membrane potential, $I_{synapse}$ - synaptic current, η - excitatory conductance, γ – inhibitory conductance, E_η - excitatory synaptic receptor reversal potential, E_γ – inhibitory synaptic reversal potential, λ_η and λ_γ – excitatory and inhibitory decay rate constants respectively.

If pre-synaptic spike arrives, either conductance η or γ increments by its maximum conductance value (aka weight), depending on a type of synapse. Consequently, magnitude of $I_{synapse}$ increases and participates in membrane dynamics (model of which can be HH, IZ, IF, or any other). At the same time, conductance starts to decay exponentially with rates λ_η or λ_γ.

With this model it is possible to reduce all synapses for a given neuron to just two: one excitatory and one inhibitory if all synapses are linear and their weights are independent of model variables [1]. Thus, whenever any synapse of a given neuron receives a spike, the model applies.

2.3. Neural network types

This section provides brief comparison of spiking neural networks and artificial neural networks.

2.3.1 Spiking neural networks

Spiking neural networks (SNNs) are type of BNNs with reduced dynamic range of transmitted signal, namely, signal at the axon terminal. Because this signal is composed of spikes, its magnitude can be approximated by zero or one in the time domain: zero when there are no spikes and one when there is a spike. Thus, the signal consists of the pulse train with variable inter-spike intervals.

Such a dramatic reduction in signal dynamic range is possible due to the properties of a synapse, signal transfer function of which can be approximated to an increase in membrane conductance followed by its exponential decay in the post-synaptic neuron as a response to arrived action potential in the pre-synaptic neuron. However, the important information about the time of arriving spike is preserved in SNNs. Membrane dynamics is also preserved in the form of membrane potential model.

Because SNNs inherit time domain from BNNs, they inherit the opportunity to model time-dependent phenomena of BNNs. Examples of phenomena are: temporal binding due to synchronized firing [26], STDP and other dependent types of plasticity, LTD, LTP, classical conditioning. Besides, axonal delay is an important temporal variable that participates in learning [27].

From another perspective, SNNs loose control over neurotransmitter release and related processes transpired in the synaptic cleft due to reduction of signal dynamic range.

2.3.2 Artificial neural networks

Typical example of neurons involved in artificial neural networks (ANNs) are perceptron (Figure 2.3-1) and ADALINE (Figure 2.3-2).

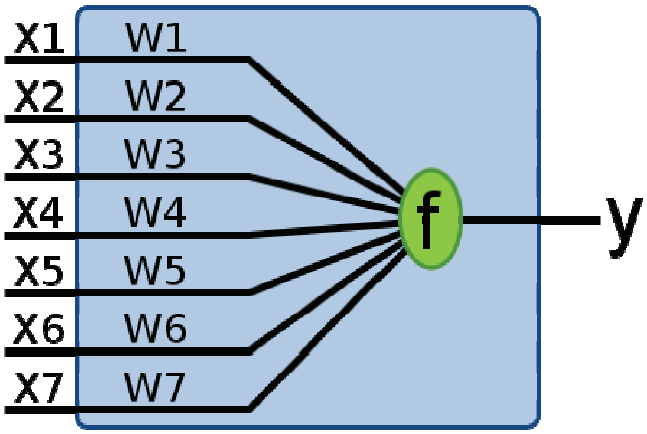


Figure 2.3-1 Simple perceptron

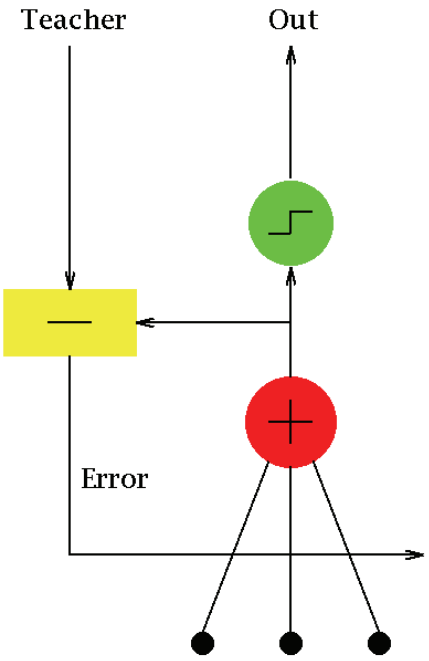


Figure 2.3-2 Simple ADALINE

Perceptron computes weighted sum of its inputs $X = \sum_{n=0}^{N} w[n]\, w[n]$ and produces an output based on the result of its activation function $Y = f(X)$. There is a variety of activation functions: step, sign, sigmoid, linear. As a rule, the result is between -1 and 1. Perceptron-based ANNs are used as classifiers. Training is required for obtaining weights. Training of multi-layered perceptron-based ANNs is usually accomplished with backpropagation algorithm, in which an error taken as difference between the output Y and desired output is propagated back in the network and used for weights ajustment [28].

ADALINE (adaptive linear neuron) is similar to perceptron with threshold or liner function. However training of ADALINE is accomplished based on the output X taken before the activation function.

After training ANNs are used in various applications: classification, estimation, filtering, and others.

Compared to SNNs, artificial neural networks (ANNs) operate on signal without relation to its time. Considering the fact that a neural signal consists of spikes, it is valid to assume that ANNs abstract signals to their steady state mean firing rate, consequently reducing the parametric space of the neuron model. This simplification, however, makes ANNs static and prohibits their application where time factor matters.

Practically all dynamic processes sensed by biological organisms and the reaction to these processes are out of ANNs application range. A few examples of such processes are sensory system of electric fish, auditory system of bats, visual system of flies, rat's position in a maze and many others [29]. Besides, ANNs lack natural dynamic learning algorithms such as STDP.

2.4. Applications of spiking neural networks

The application domain of SNNs is diverse: data classification and pattern recognition (visual and sound recognition, speech processing, image feature detection and others), prediction and estimation (market predictions, signal processing, adaptive

filtering and others), control (control systems, robotics and others), prosthetics (silicon retina, artificial cochlea and others), neurological and neuroscientific modeling, cryptography, and others [30].

In this section three applications of SNNs are reviewed from the following domains: image processing, signal processing, and robotics.

2.4.1 Image processing

There are a variety of SNN application in image and visual processing. The majority of them are based on networks with topologies and properties taken from visual cortex of animals and humans. Due to the highly ordered structure of visual cortex it provides benefits of network computation allocation and allows easy optimization of computing architectures designed for representation of visual cortex using SNNs.

An example of SNN applied to edge detection based on conductance-based integrate-and-fire neuron model is demonstrated in [31]. Multi-layered SNN based on receptive fields of visual cortex is proposed (Figure 2.4-1): 1) receptor layer responsible for translating pixel gray scale value to neuron conductance, 2) an intermediate layer that detects edges based on spatial orientation of dark-light boundary in respect to the synaptic connections of this layer with receptor layer and the type of connections, 3) output layer that integrates inputs from intermediate layer and creates an image of detected edges in the form of spiking rates.

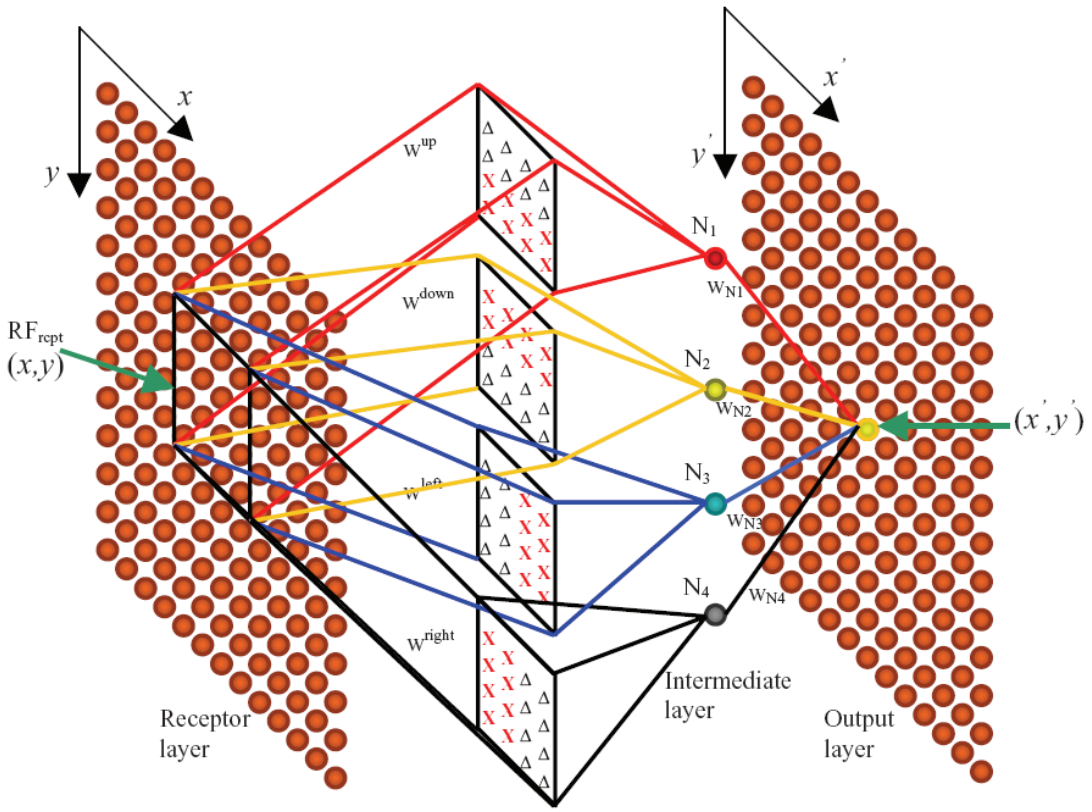


Figure 2.4-1 Spiking Neural Network Model for Edge Detection [31]

Within square detection area neurons in the intermediate layer produce spikes only if excitatory synapses (X) have a higher activity than inhibitory synapses (Δ), which corresponds to the case when lighter image area overlaps excitatory part of square detection area and darker image area overlaps inhibitory part. Adjusting synaptic weight matrices allows controlling sensitivity of the system to the gray scale value of the edge and can be applied for attention modeling based on spatial-dependent synaptic weight control (Figure 2.4-2). The size of square detection area allows controlling granularity of edge detection.

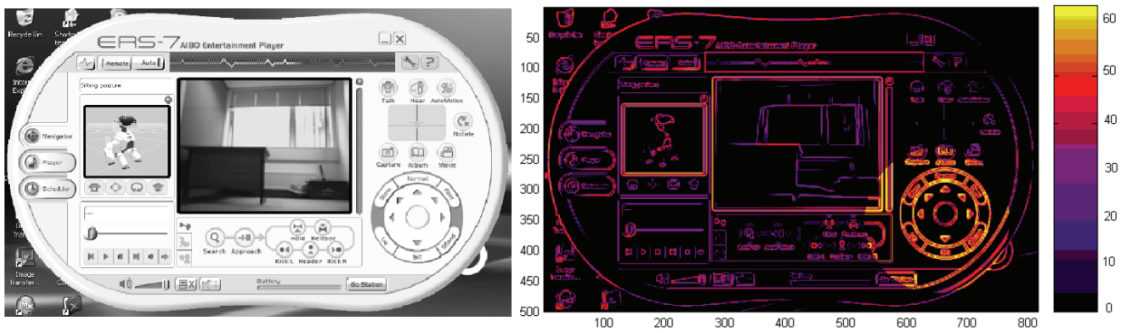


Figure 2.4-2 Image and its firing rate representation with attention area produced by edge detecting SNN [31]

2.4.2 Signal processing

Signal processing is another area of SNN application. With the use of SNN, designed according to BNN of an auditory system, it is possible to encode a signal using spikes and reconstruct the signal with SNR high enough to emphasize similarity of both sampled and reconstructed signals. Natural auditory system utilizes narrow-band signal decomposition with low-frequency spike encoding within each band.

An example of non-linear signal processing system with SNN is shown in [32]. Short Time Fourier Transform is applied in order to obtain narrow-band signal decomposition. Detailed steps of this process are as following:

Continuous-Time Fourier Transform provides ideal signal spectrum representation:

$$X_c(jw) = \int_{-\infty}^{\infty} x_c(t) e^{-jwt} dt \qquad (2.4\text{-}1)$$

Discrete Fourier Transform approximates an integral with finite sum (sampled signal), evaluates signal at discrete frequencies within finite time interval:

$$\hat{X}_c(jw_k) = \sum_{n=0}^{N-1} x_c\,[n]\; e^{-jw_k n\, T_s}\; T_s;\ \mathrm{k} = 0,1,\dots,\mathrm{N}-1;\ w_k = \frac{2\pi k}{N\, T_s} \qquad (2.4\text{-}2)$$

$$X[k] = \sum_{n=0}^{N-1} x\,[n]\; e^{-j(2\pi/N)kn};\ \mathrm{k} = 0,1,\dots,\mathrm{N}-1 \qquad (2.4\text{-}3)$$

Windowing limits signal sequence in time and band-limits signal frequency:

$$X[k] = \sum_{n=0}^{N-1} w[n]\; x\,[n]\; e^{-j(2\pi/N)kn};\ \mathrm{k} = 0,1,\dots,\mathrm{N}-1 \qquad (2.4\text{-}4)$$

Type of the window function defines Q-factor of windowing filter used in narrow-band decomposition (Figure 2.4-3):

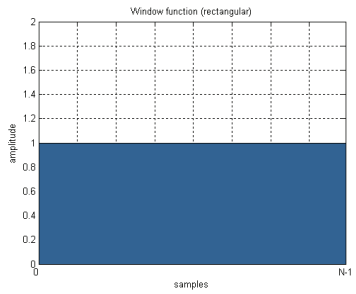
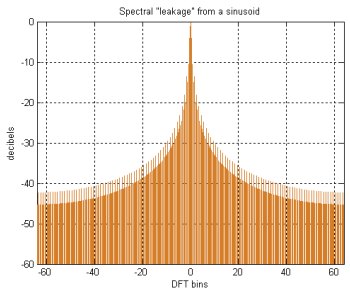
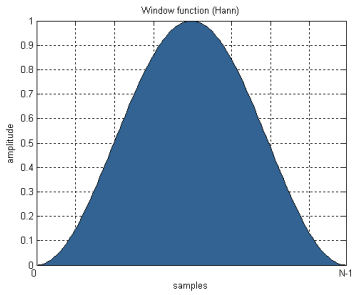
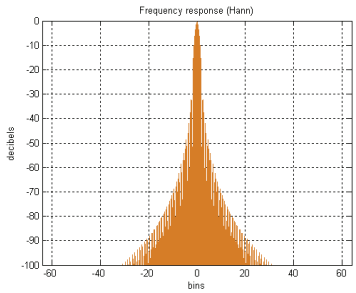
Rectangular	  $w[n] = 1$ (2.4-5)
Hann	  $w[n] = 0.5\left(1 + cos\left(\frac{2\pi n}{N-1}\right)\right)$ (2.4-6)

Figure 2.4-3 Rectangular and Hann windows: time and frequency domains

Frame-based narrow-band spectral decomposition of signal allows extracting spectrum for each frame l and subband k of sampled signal $x[lH + n]$ by scaling it with window filter $w[n]$:

$$X_l[k] = \sum_{n=0}^{N-1} w[n]\, x\,[lH + n]\, e^{-j\left(\frac{2\pi}{N}\right)kn} \qquad (2.4\text{-}7)$$

$$k = 0,1, \dots, N-1; l = 0,1, \dots, F; H < N$$

Band-limited by narrow band window signal can be synthesized from each subband spectral representation using inverse DFT and summing over all overlapping frames for each signal component n:

$$x_k[n] = \sum_{l}^{\forall\, overlapping\ l} \left(\frac{1}{N} \sum_{k=0}^{N-1} X_l[k]\, e^{-j\left(\frac{2\pi}{N}\right)k(n\%H)} \right);\ n = 0,1, \dots, N-1 \qquad (2.4\text{-}8)$$

As a result, N split-by-band signal sequences are obtained, each of which can be viewed as an amplitude-modulated signal with frequency band around carrier frequency defined by the center frequency of window filter and with band size defined by that of window filter. The process of spectral decomposition is similar to narrow band filtering by the mechanical vibrations of the basilar membrane in the inner ear followed by transduction by the hair cells. The next step is coding of each band-limited signal as a spike trains generated by cochlea neurons.

One of the most important aspects of this coding includes refractory after-spike period. During the refractory period a neuron is not able to generate new spike until after voltage-gated Na^+ channels deinactivate (see section 2.1.1 for details). The refractory period can be modeled using exponential decay function between current time t and previous spike time t_{i-1}:

$$g(t) = Ae^{\left(-\frac{t-t_{i-1}}{\tau}\right)} \quad (2.4\text{-}9)$$

As a result, signal $x_k[n]$ is encoded with spike times $t_k[s]$, with $s = 0,1,\dots,Y-1, Y < N$ (Figure 2.4-4).

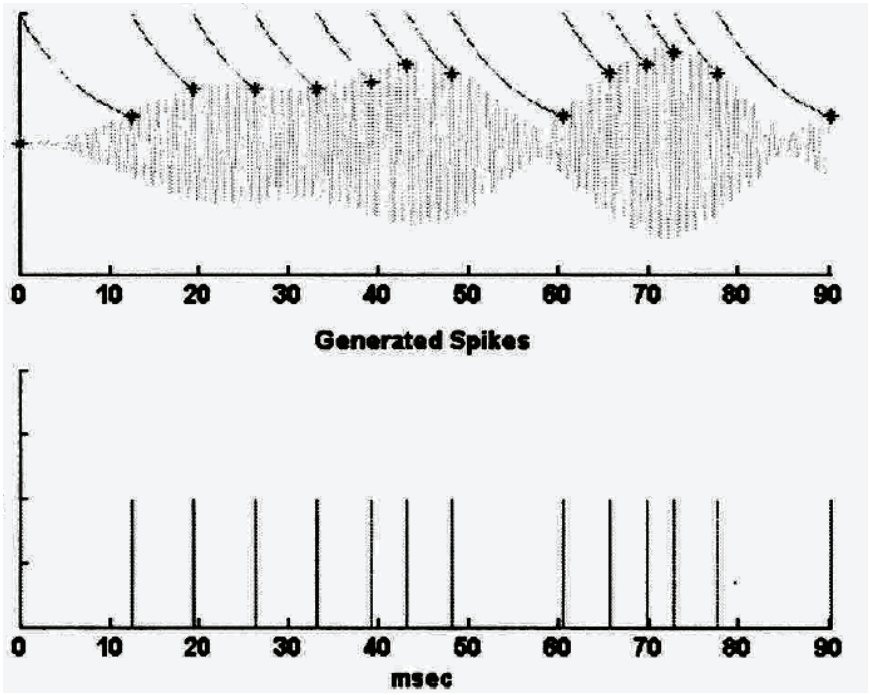


Figure 2.4-4 Spike encoding of signal for one of the subbands [32]

Encoded signal can be used for audio pattern recognition. Reconstruction of signal is possible as well since modulating function (2.4-9) is known. With the assumption that each narrow band signal is sampled at the peak, the amplitude and the phase can be calculated using modulating function and complex sinusoid approximation. As a result of these operations, simple auditory signal such as speech can be sampled, encoded, decoded and reconstructed (Figure 2.4-5).

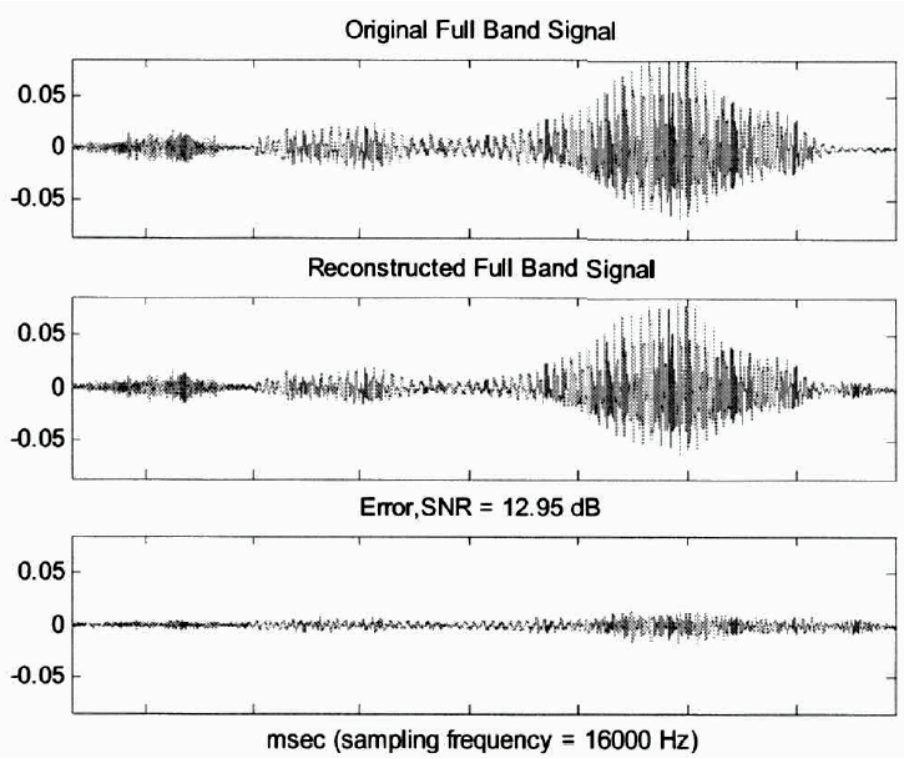


Figure 2.4-5 Spike encoding of signal for one of the subbands [32]

2.4.3 Robotics

Robotics is one of the most utilized areas of SNN application. An example of robot that learns how to avoid obstacles and how to find targets based on conditioned stimulus and STDP is reported in [33] (Figure 2.4-6).

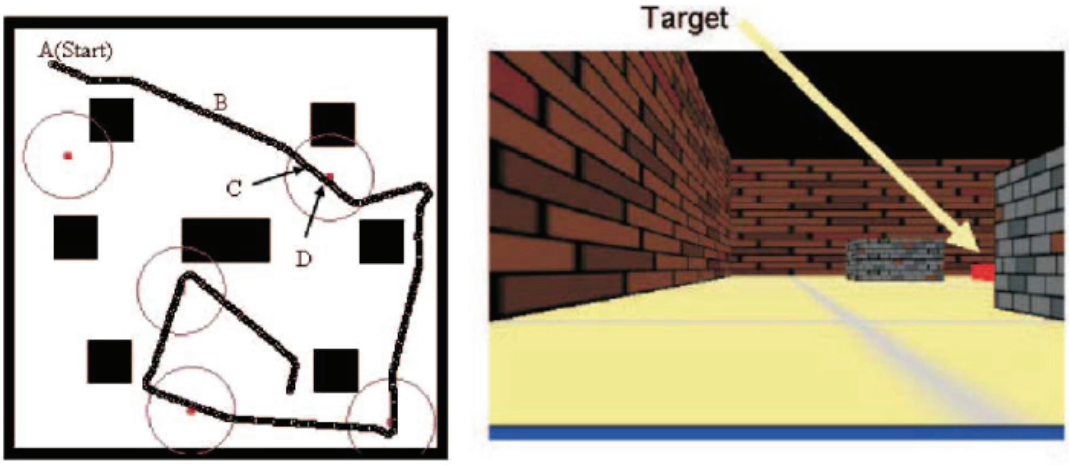


Figure 2.4-6 Obstacle avoidance and target search in robotics [33]

Following the rules of classical conditioning described in section 2.1.3, stimuli are classified in this application as: 1) Unconditioned stimuli/reflex: encountering an obstacle with immediate reflexive withdrawal as a response. 2) Conditioned stimuli/reflex: in case of an obstacle avoidance they can be received by long distance range perception sensors. Conditioned response is to avoid an obstacle from the distance. If both encountering an obstacle and sensing it from the distance are presented to a robot at the same time, eventually its SNN learns how to avoid an obstacle without encountering it.

Underlying mechanism of classical conditioning is STDP and synaptic plasticity in general (see section 2.1.3). In this example an SNN with Izhikevich neurons [3] and STDP rules for synaptic connections from conditioned sensor neurons has been developed (Figure 2.4-7).

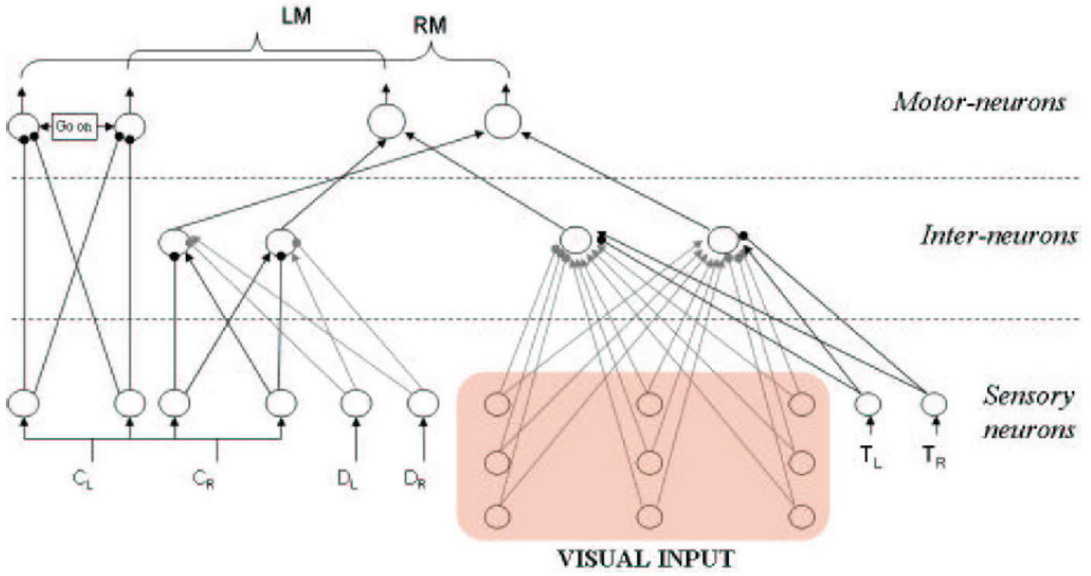


Figure 2.4-7 SNN topology for obstacle avoidance and target search in robotics [33]

In Figure 2.4-7 unconditioned stimuli of obstacle and target encountering are presented to sensor neurons C_L, C_R and T_L, T_R respectfully. Conditioned stimuli of an obstacle and target are presented to D_L, D_R and visual input sensory neurons respectfully. The latter mapped to receptive fields of an image produced by a camera. Conditioned neurons have plastic synapses that target inter-neurons (gray) and develop according to the following STDP rule for a synaptic weight *w*:

$$\Delta w = \begin{cases} A_+ e^{\left(\frac{\Delta t}{\tau_+}\right)}, & if\ \Delta t < 0\ (potentiation) \\ A_- e^{\left(\frac{-\Delta t}{\tau_-}\right)}, & if\ \Delta t \geq 0\ (depression) \end{cases}, \ \Delta t = t_{pre} - t_{post} \tag{2.1.5.10}$$

$$w(t+1) = \begin{cases} 0.95\, w(t) + \Delta w, & if\ t\%3000 = 0 \\ w(t) + \Delta w, & otherwise \end{cases} \tag{2.1.5.11}$$

In case of an obstacle both types of sensor neurons (C_L, C_R and D_L, D_R) produce spikes when the obstacle is encountered and, according to the STDP rules D_L, D_R increase the strength of their synaptic connections with inter-neurons. After reaching a certain value synapses of D_L and D_R are strong enough to transfer the signal from these neurons

without help of C_L and C_R, and the robot acquires the ability to avoid obstacles. Similar learning process occurs in a case of target search except that the motor response of the robot is to reach the target as soon as it is detected. Comparison of number of detected targets for trained and not trained robots is presented in Figure 2.4-8.

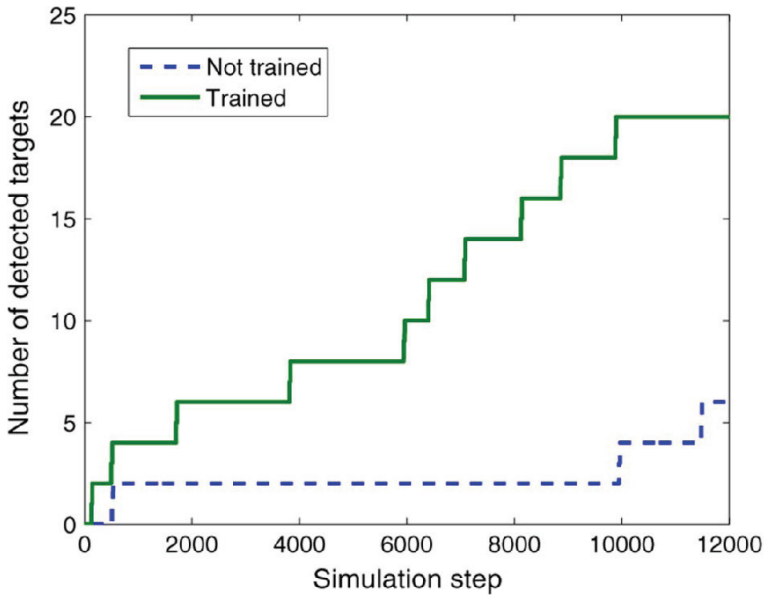


Figure 2.4-8 Comparison of number of detected targets [33]

Chapter 3 System modeling approaches

A system of functioning SNN can be characterized from the perspective of time as: synchronous, asynchronous, or hybrid. There is a variety of system reduction and optimization techniques. An example of such can be synaptic reduction for linear and additive synapses [1], compartmental to single compartment reduction.

In this section simple non-reduced systems are reviewed from the perspective of time. In addition, some numerical integration techniques and synaptic data structure representations are presented due to their importance in system modeling.

3.1. System types: synchronous, asynchronous and hybrid

In a synchronous (clock driven) system model state variable updates (update phase) and spike communication (propagation phase) are accomplished with every integration step (Figure 3.1-1). The integration step is constant.

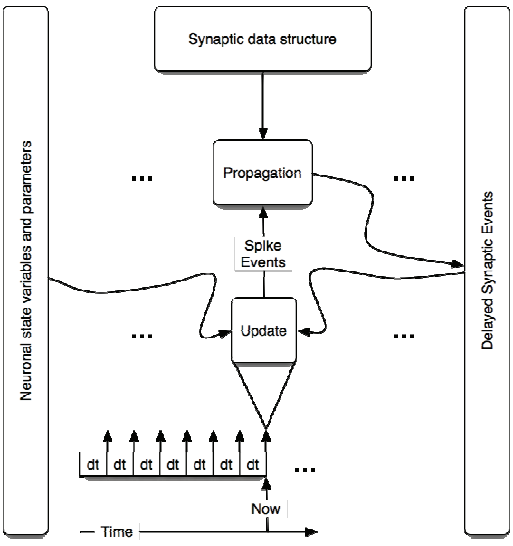


Figure 3.1-1 Synchronous system: simulation execution flow

The order of computation for this type of systems per second of simulation time is defined by the following equation [1]:

$$O_{update}\left(\frac{N}{dt}\right) + O_{propagation}(FNp) \quad (3.1\text{-}1)$$

where N is network size, F is average firing rate, p is average target neurons per spike source.

Dominating phase (update or propagation) is determined by complexity of a neuron model, density of synaptic connections and complexity of the algorithm responsible for handling delays. For this type of systems delays can be effectively managed by circular buffers: fixed time resolution dt determines both update and propagation delay time steps.

Major drawbacks of the synchronous system are: 1) Spike events are tight to time grid, which is defined by dt. This quantization error propagates throughout the simulation, which is aggressively expressed on a network level. Thus, the precision of a computation depends on the size of dt: the smaller the size the higher the precision, but the more computation per unit of simulation time (Figure 3.1-2). 2) There is a high probability that the event of membrane potential crossing spike threshold can be missed since the test for the threshold is done at fixed points in time. 3) Delay time resolution is determined by dt.

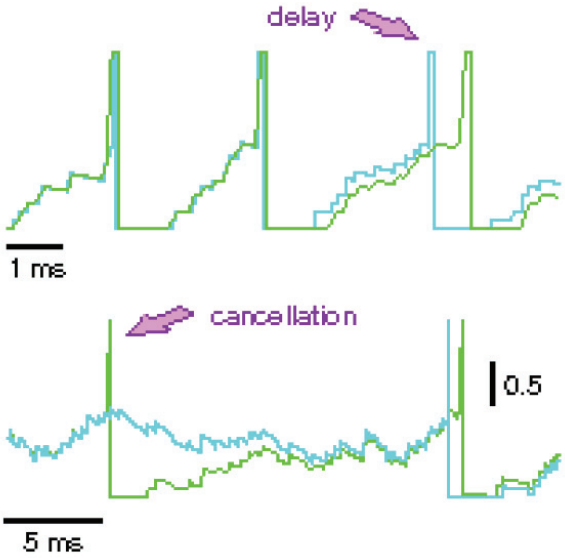


Figure 3.1-2 Delays and cancellations due to quantization error in a synchronous system [1]

In asynchronous (event driven) system (Figure 3.1-3) model variable updates and spike communication are done for the next event to execute in the sorted event queue, be it a spike or a synaptic event. If it is a spike, than its source neuron state is updated, neuron's future spike time is calculated and inserted in the queue and all synaptic events, resulting from the current spike, are inserted in the queue as well according to individual delays. If the next event is a synaptic event, the state variable of its target neuron is updated, and neuron's future spike time is recalculated and relocated within the queue according to its new value.

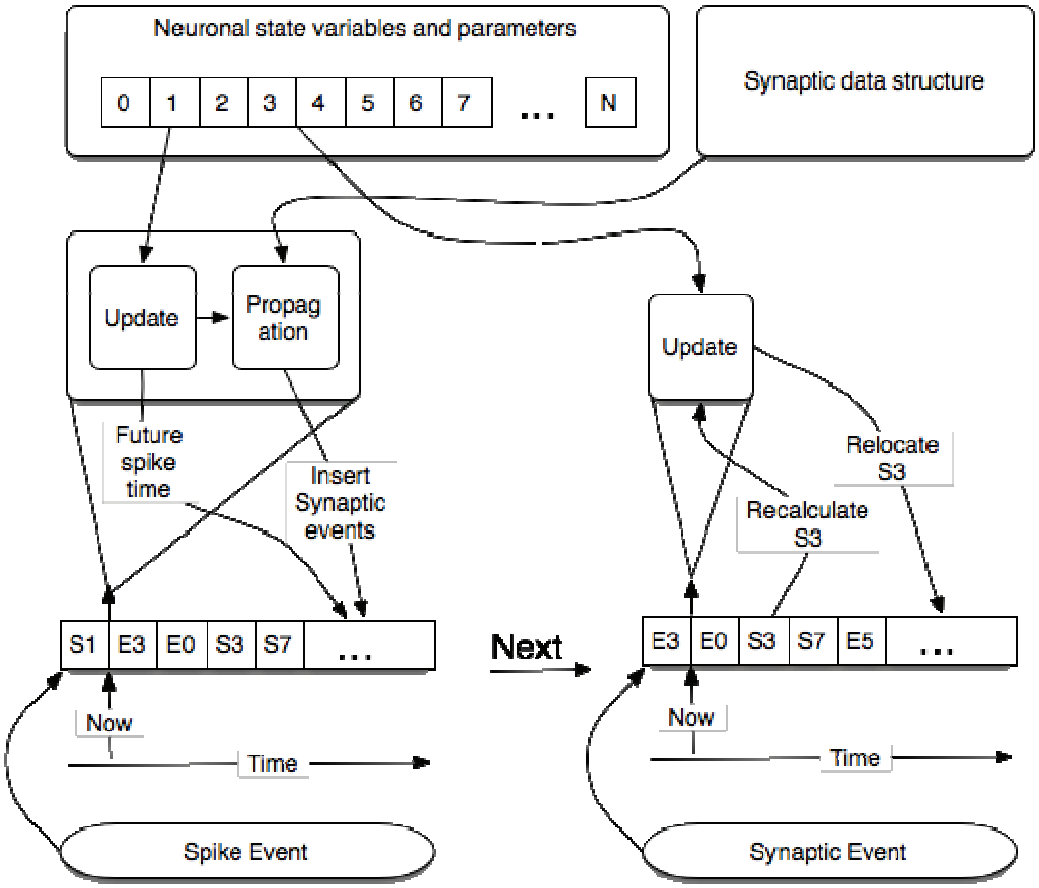


Figure 3.1-3 Asynchronous system: simulation execution flow

Computation order for asynchronous system per second of simulation time is stipulated by the following formulae [1]:

$$O(FNp) \tag{3.1-2}$$

This equation describes that within each simulated second N neurons produce FN spikes and distribute these spikes to p synapses. Although the computation order is less than that defined by equation (3.1-1), execution time of a single operation is likely to be

more than that of propagation operation in (3.1-1), because it includes update and propagate execution times. Besides, if implemented in software, data structure handling (insert, extract, sort, shift etc) is a crucial component of such an operation and may result in memory bounded implementations. This handling can be somewhat reduced if transmission delays are guaranteed to be more than certain value. Indeed, in this case all events within this time starting from the current time are certified not to be changed.

The benefit and, at the same time, the drawback of asynchronous system is independence of its time flow from dt, which requires explicit solutions for neuron model differential equations in order to calculate model variable value at any time. However, if such solutions do not exist, event-driven updates of the state variables can result in accuracy degradation due to sparsely spaced in time evaluations of state variables if future spike times do not exist. Nevertheless, this depends on numerical integration method chosen (discussed further) as well as how it is applied within the system. Conversely, the accuracy of asynchronous system outperforms that of synchronous system, because of precise event timing, if there are enough model variable updates per unit time for all neurons in the network. In the latter case and with software implementations this accuracy can be as high as the precision of data types used (single precision FP, double precision FP, etc). This is important for systems targeting implementation of high-accurate biological mechanisms, such as STDP (see section 2.1.3).

An example of comparison event-driven and clock-driven simulation of a system with STDP is presented in Figure 3.1-4. In this figure in case of clock-driven system all three synapses are facilitated since the system associates producing new spike with all three events. However, in the case of event driven system only first two synapses are facilitated, but the third one is depressed since it follows post-synaptic action potential.

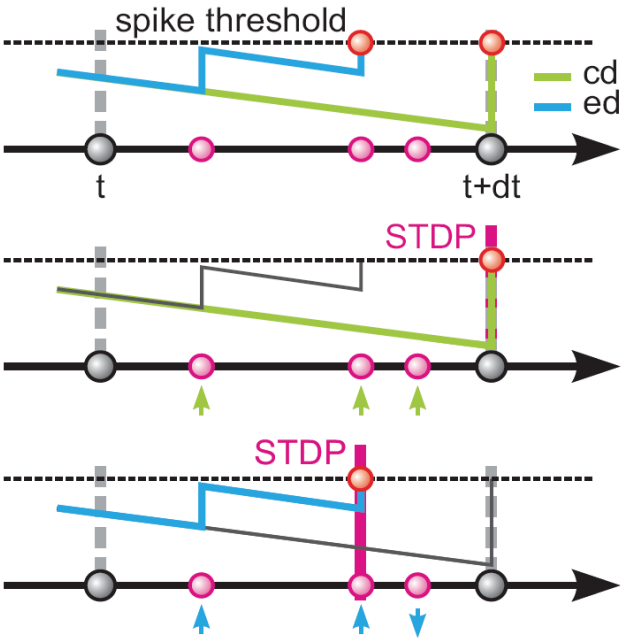


Figure 3.1-4 Dynamics in neuronal systems with STDP: impact of the simulation strategy (clock-driven: cd; event-driven: ed) on the facilitation and depression of synapses [1]

Event-driven systems have computational advantage in speed due to applying computation only at the time of the event. However, this advantage may be circumvented in parallel implementation by the fact that event-driven system serializes spike events in respect to time, which prohibits processing their target synapses in parallel. For example, if in a case of synchronous system two synaptic events are scheduled to execute at the same integration step, then in a case of asynchronous system even a slight deviation in time of these events results in serialization of their processing.

The effect of low accuracy of the system on temporal dynamics of state variables in a neural network depends on the features implemented in that network. If such features require high-accurate timing, STDP for example, then in order to accommodate the required precision, an integration step has to be small enough in case of synchronous system and there should be enough state variable updates per unit time in case of asynchronous system (specific to the integration method used). Hybrid systems attempt to resolve this issue.

A hybrid system (Figure 3.1-5) is a merged clock-and-event driven system. In this type of system state variable updates and spike communication are done with every integration step. However, time precision of spike and synaptic events is not limited by the time grid. Instead, events are recorded in the queue following the principle of asynchronous system. The size of time step is determined by the number of updates per unit of simulation time necessary to guarantee the precision for neurons with a low number of events.

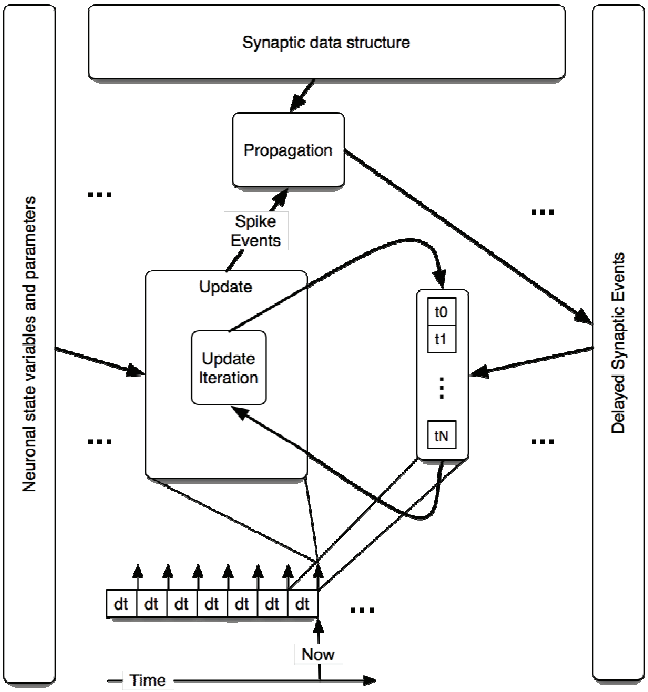


Figure 3.1-5 Hybrid system: simulation execution flow

3.2. Numerical integration techniques

Numerical integration methods are useful for obtaining numerical values of solutions to differential equation with an initial value problem (equation (3.2-1)) in cases when analytical solution is not possible and in computer-based solutions.

$$y'(t) = \frac{dy}{dt} = f\big(t, y(t)\big), \qquad y(t_0) = y_0, \qquad t \in [t_0 - \alpha, t_0 + \alpha] \tag{3.2-1}$$

Methods can be classified as multistep (involve several past values in order to obtain the next one) and single step (involve only one past value in order to obtain the next one). As a rule, single step methods require many more evaluations of derivative in order to approach their accuracy to that of multistep methods [34]. In this section several methods are reviewed.

3.2.1 Euler Method

Euler method is the simplest single step method. It has limited application due to its low accuracy [34], and is used mostly for demonstration purposes. The method is based on tangential approximation of solution to IVP (3.2-1) at discrete points in time with the interval h:

$$y_{n+1} = y_n + hf(t_n, y_n) \tag{3.2-2}$$

As a consequence, the method introduces error, especially on the intervals where $\frac{dy}{dt}$ changes rapidly. The error is proportional to $O(h^2)$. If accumulated over some period of simulation time, it eventually results in complete difference of solution compared to the reference solution (Figure 3.2-1).

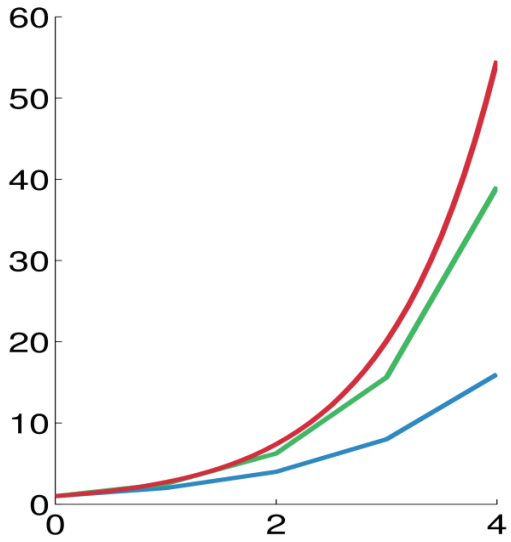


Figure 3.2-1 Illustration of numerical integration for the equation y' = y, y(0) = 1. Blue: the Euler method, green: the midpoint method, red: the exact solution, $y = e^t$. The step size is h = 1.0.

Modified Euler method provides better accuracy. It uses the average slope between two points on y, where the second point is determined by the original Euler method. Thus, this method belongs to a predictor-corrector class of methods. It can provide the desired number of approximations (order) at any point:

$$y_{n+1}^{(i+1)} = y_n + \frac{h}{2}\left(f(x_n, y_n) + f\left(x_{n+1}, y_n^{(i)}\right)\right), \qquad i = 0,1,2, \ldots \tag{3.2-3}$$

Given initially: $y_{n+1}^{(1)} = y_n + hf(t_n, y_n)$

One step of modified Euler method provides an error of $O(h^3)$. Successive steps may give better accuracy. However, for steep functions with large Lipschitz constants, the method still requires a small step. Other predictor-corrector methods such as Milne's Simpson Method and Adams-Moulton produce an error on the order of $O(h^5)$

3.2.2 Runge-Kutta 4th order method

Runge-Kutta methods agree with Taylor series method, which is considered a standard (but computationally impractical) since it represents solution function as Taylor series expansion, up to terms of h^r, where *r* is the order of RK method. Thus, for RK4 the increment error starts at h^5. RK methods are single step methods. In particular, RK4 is widely used for numerical solutions of IVPs. However, the major drawback of RK methods is utilization of several evaluations of $\frac{dy}{dt}$ depending on the order. For RK4, the step for an IVP with a single equation is:

$$
\begin{aligned}
k_1 &= hf(x_n, y_n) \\
k_2 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) \\
k_3 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right) \\
k_4 &= hf(x_n + h, y_n + k_3) \\
\Delta y &= \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\
x_{n+1} &= x_n + h, \qquad y_{n+1} = y_n + \Delta y
\end{aligned}
\tag{3.2-4}
$$

3.2.3 Bulirsch–Stoer method

Bulirsch-Stoer method uses modified midpoint method with evaluation and error tolerance check using extrapolation with rational functions. In [35] it is mentioned as “the best-known way to obtain high accuracy solutions to ordinary differential equations with minimal computational effort.” However, it is better suited for smooth functions with smaller Lipschitz constants, which is sometimes not the case for neuron models. The method procedure is the following:

1) Compute solution of y_{n+1} at x_{n+1} using modified midpoint method:

$$
\begin{aligned}
&z_0 = y_n \\
&z_1 = z_0 + hf(x_n, z_0) \\
&z_{m+1} = z_{m-1} + 2hf(x_n + mh, z_m), \qquad m = 1, 2, \dots, k-1 \\
&y_{n+1} = \frac{1}{2}(z_k + z_{k-1} + hf(x_n + H, z_k)) \\
&x_{n+1} = x_n + H, \qquad h = \frac{H}{k}, \qquad h_j = \frac{H}{k_j} \\
&k = 2, 4, 6, 8, \dots \;\; k_j = 2k_{j-2}
\end{aligned}
\tag{3.2-5}
$$

where H is a step size, k is a number of substeps within H

2) Fit the solution to analytic form and evaluate at $h = 0$ using extrapolation with Aitkens-Neville algorithm for each successive k until error tolerance is satisfied:

$$
\begin{aligned}
&T_{11} \\
&T_{21} \quad T_{22} \\
&T_{31} \quad T_{32} \quad T_{33} \\
&\dots \quad \dots \quad \dots \quad T_{ll} \\
&T_{j,1}(h) = y_{n+1}^{k_j} \\
&T_{j,l}(h) = y_{n+1}^{k_j} \frac{(h - h_{j-l+1})T_{j,l-1}(h) - (h - h_j)T_{j-1,l-1}(h)}{h_j - h_{j-l+1}} \\
&\left\| T_{l+1,l} - T_{l+1,l+1} \right\|^1 < \epsilon
\end{aligned}
\tag{3.2-6}
$$

3) Use last $T_{j,l}(0)$ for solution.

3.2.4 Parker-Sochacki

Parker-Sochacki (PS) [36] numerical integration technique is based on application of Maclaurin series properties to the solution of differential equations. The method was originally developed based on Picard iteration [37]. Compared to the methods described above, PS method provides an adaptive order for the given error tolerance. The order depends on Lipschitz constant of the solution function at the point in time, when the method is applied.

Picard iteration is the basis of Picard–Lindelöf theorem. The theorem states that if initial value problem (3.2-1) is Lipschitz continuous in y and continuous in t, then for some value $\varepsilon > 0$, there exists a unique solution $y(t)$ to the initial value problem (IVP) within the range $[t_0 - \varepsilon, t_0 + \varepsilon]$, [38].

The theorem can be proven by showing convergence of sequence of Picard iterates and finding the exact solution at t as limit of this sequence:

$$\varphi_0(t) = y_0, \qquad \varphi_i(t) = y_0 + \int_{t_0}^{t} f\big(s, \varphi_{i-1}(s)\big)\, ds\,, \qquad y(t) = \lim_{i\to\infty} \varphi_i(t) \tag{3.2-7}$$

The requirements of IVP to be Lipschitz continuous is local, that is:

$$f: T \to Y, \qquad d_Y\big(f(t_1), f(t_2)\big) \le K d_T(t_1, t_2), \qquad t_1 \in [t_0 - \alpha, t_0 + \alpha], \quad t_2 \in [t_0 - \alpha, t_0 + \alpha] \tag{3.2-8}$$

and therefore, φ can be described with Taylor series expansion since power functions are locally Lipschitz continuous. However, such a description results in increasing complexity due to successive integration. Instead of using Picard iteration

directly, PS method approaches to the solution of IVP from the perspective of Maclaurin series properties. Given that y is Lipschitz continuous it can be described as:

$$y(t) = \sum_{p=0}^{\infty} y_p t^p , \qquad y_p = \frac{y^{(p)}(0)}{p!} \tag{3.2-9}$$

and its derivative can be described as:

$$y'(t) = \sum_{p=0}^{\infty} y'_p t^p , \qquad y'_p = \frac{y^{(p+1)}(0)}{p!} \tag{3.2-10}$$

after substituting (3.2-9) in (3.2-10) $y'(t)$ can be described as:

$$y'(t) = \sum_{p=0}^{\infty} y'_p t^p = \sum_{p=0}^{\infty} \frac{y^{(p+1)}(0)}{p!} t^p = \sum_{p=0}^{\infty} (p+1) \frac{y^{(p+1)}(0)}{(p+1)!} t^p = \\ = \sum_{p=0}^{\infty} (p+1) y_{p+1} t^p \tag{3.2-11}$$

Thus, each consecutive coefficient of power series can be obtained based on derivative of previous coefficient:

$$y_{p+1} = y'_p/(p+1) \tag{3.2-12}$$

This expression is useful in eliminating derivative in IVP (3.2-1):

$$y'(t) = \sum_{p=0}^{\infty} (p+1) y_{p+1} t^p = f\left(t, \sum_{p=0}^{\infty} y_p t^p\right) \tag{3.2-13}$$

However, the problem becomes specific to the form of $f\big(t, y(t)\big)$. For example, provided that $f\big(t, y(t)\big) = ky(t) + b$ is linear or shifted for convenience in order to eliminate constant term: $\overline{f\big(t, y(t)\big)} = f\big(t, y(t)\big) - b$, the IVP (3.2-1) expands to:

$$\sum_{p=0}^{\infty} (p+1) y_{p+1} t^p = k\left(\sum_{p=0}^{\infty} y_p t^p\right) \tag{3.2-14}$$

With finite order N instead of ∞ the IVP becomes solvable iteratively, which is suited for computer-based solution:

$$\begin{aligned}
&y = y_0;\ \ t = 1;\\
&FOR:\ \ p = 0, \qquad p < N, \qquad p{+}{+}:\\
&\quad y_p = k y_p/(p+1);\\
&\quad y = y + y_p t;\\
&\quad t = t \times \Delta t;\\
&END\\
&y = y + b;
\end{aligned} \tag{3.2-15}$$

Solution (3.2-15) exhibits loop level parallelism (LLP) and parallel reduction, which can be exploited if all coefficients are pre-calculated.

With $f\big(t,y(t)\big) = ay^2(t) + by(t) + c$ the IVP (3.2-1) has a quadratic term. With shifted form, $\overline{f\big(t,y(t)\big)} = ay^2(t) + by(t)$, it expands to:

$$\sum_{p=0}^{\infty}(p+1)y_{p+1}t^p = a\left(\sum_{p=0}^{\infty}y_pt^p\right)\left(\sum_{p=0}^{\infty}y_pt^p\right) + b\left(\sum_{p=0}^{\infty}y_pt^p\right) \tag{3.2-16}$$

The quadratic term can be readily converted to more convenient form with series multiplication:

$$\begin{aligned}
&\left(\sum_{p=0}^{\infty}y_pt^p\right)\left(\sum_{p=0}^{\infty}y_pt^p\right) = \\
&= \frac{\begin{array}{rcl} & & \times\ y_0 + y_1t + y_2t^2 \ldots \\ & & y_0 + y_1t + y_2t^2 \ldots \end{array}}{y_0^2 + \sum_{i=0}^{1}y_iy_{1-i}t + \sum_{i=0}^{2}y_iy_{2-i}t^2 + \cdots + \sum_{i=0}^{p}y_iy_{p-i}t^p + \cdots} = \\
&= \sum_{p=0}^{\infty}\left(\sum_{i=0}^{p}y_iy_{p-i}\right)t^p
\end{aligned} \tag{3.2-17}$$

the result is an infinite sum of convolutions (Cauchy products). As a consequence, (3.2-16) can be written as:

$$\sum_{p=0}^{\infty}(p+1)y_{p+1}t^p = a\sum_{p=0}^{\infty}\left(\sum_{i=0}^{p}y_i y_{p-i}\right)t^p + b\left(\sum_{p=0}^{\infty}y_p t^p\right) \tag{3.2-18}$$

With finite order N the pseudo code for solving IVP becomes:

$$\begin{aligned}
&y = y_0;\ \ t = 1;\\
&FOR:\ \ p = 0, \qquad p < N, \qquad p++:\\
&\quad d = 0;\\
&\quad FOR:\ \ i = 0, \qquad i \le p, \qquad i++:\\
&\qquad d = d + y_i y_{p-i};\\
&\quad END\\
&\quad y_p = (ad + by_p)/(p+1);\\
&\quad y = y + y_p t;\\
&\quad t = t \times \Delta t\\
&END\\
&y = y + c;
\end{aligned} \tag{3.2-19}$$

Exploiting parallel computation is problematic in this case because of a linearly scaled convolution and loop-carried circular dependence on d. Partial parallelism still can be exploited in a case of convolution and in a case of $(by_p)/(p+1)$.

As it can be seen, for IVPs with higher power terms iterative convolutions increase computational complexity. However, other techniques for power term computation that can be incorporated into PS method exist [39], but they still use at least one convolution.

Based on the fact that the solution of ODE can be expanded with Maclaurin series under certain conditions, PS method brings that expansion into original ODE, which allows representing ODE as an equation operating on infinite or finite sums, where the

number of summands defines solution accuracy. PS method also can be used for the systems of equations. Parallel techniques can be applied with PS method. However, Cauchy products that appear as a result of power operations tend to reduce parallelism.

3.3. Synaptic data structures

The goal of synaptic data structures regardless of methods of their implementation is storing and accessing synaptic connectivity data in the most efficient way.

In its most simplistic representation synaptic connectivity data structure is a matrix. Indeed, with the size of NxN, where N is an SNN size, each row of this matrix defines target neuron data (weights and delays) for a source neuron with this row number. Thus, the matrix is bounded by the largest synaptic density per source neuron. If density varies significantly from source to source then matrix is most likely sparse.

There are several implementations of sparse matrices: diagonal, ELLPACK, coordinate, compressed sparse row (CSR), hybrid, packet, and other formats [40].

Diagonal format is designed for representation of diagonal-like matrices (Figure 3.3-1). Data and offset from the main diagonal are stored in separate matrices. Diagonal matrices have a specific topology and are rarely used in synaptic data structures with arbitrary topologies.

$$\mathtt{A} = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix} \rightarrow \begin{array}{r} \mathtt{data} = \begin{bmatrix} * & 1 & 7 \\ * & 2 & 8 \\ 5 & 3 & 9 \\ 6 & 4 & * \end{bmatrix} \\ \mathtt{offsets} = \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \end{array}$$

Figure 3.3-1 Diagonal format of sparse matrix representation [40]

ELLPACK [41] is more storage-efficient for matrices with uniform number of non-zero columns or rows (Figure 3.3-2). Zeroes are extracted from original matrix and

data is packed into rows. As a result, data matrix width is defined by the largest row of non-zero elements in the original matrix. Indices define position of compressed elements relative to the original matrix. This format is suitable for SNN with uniform synaptic density.

$$\mathtt{A} = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix} \rightarrow \begin{matrix} \mathtt{data} = \begin{bmatrix} 1 & 7 & * \\ 2 & 8 & * \\ 5 & 3 & 9 \\ 6 & 4 & * \end{bmatrix} \\ \mathtt{indices} = \begin{bmatrix} 0 & 1 & * \\ 1 & 2 & * \\ 0 & 2 & 3 \\ 1 & 3 & * \end{bmatrix} \end{matrix}$$

Figure 3.3-2 ELLPACK format of sparse matrix representation [40]

Coordinate format is a general sparse-matrix representation format based on column and row indices (Figure 3.3-3). No padding is necessary compared to ELLPACK. However, for each data element there are two coordinate numbers required: row and column. This format is suitable for SNN with any topology.

$$\mathtt{A} = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix} \rightarrow \begin{matrix} \mathtt{row} = \begin{bmatrix} 0 & 0 & 1 & 1 & 2 & 2 & 2 & 3 & 3 \end{bmatrix} \\ \mathtt{col} = \begin{bmatrix} 0 & 1 & 1 & 2 & 0 & 2 & 3 & 1 & 3 \end{bmatrix} \\ \mathtt{data} = \begin{bmatrix} 1 & 7 & 2 & 8 & 5 & 3 & 9 & 6 & 4 \end{bmatrix} \end{matrix}$$

Figure 3.3-3 Coordinate format of sparse matrix representation [40]

CSR is a modified version of coordinate format where either row or column coordinate array is replaced with a pointer to starting position of this row or column in data matrix. For example, in Figure 3.3-4 when extracting a number from the 3rd row and 3rd column (position (2, 2)) pointer 4 is obtained, which gives the base address for this row in the data and indices matrices. From there, searching for column number 2 until

found or until difference between two pointers is reached results in extraction of data element with value equal 3. This format requires less storage compared to coordinate format and is efficient for SNN implementation.

$$\mathtt{A} = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix} \rightarrow \begin{aligned} \mathtt{ptr} &= \begin{bmatrix} 0 & 2 & 4 & 7 & 9 \end{bmatrix} \\ \mathtt{indices} &= \begin{bmatrix} 0 & 1 & 1 & 2 & 0 & 2 & 3 & 1 & 3 \end{bmatrix} \\ \mathtt{data} &= \begin{bmatrix} 1 & 7 & 2 & 8 & 5 & 3 & 9 & 6 & 4 \end{bmatrix} \end{aligned}$$

Figure 3.3-4 CSR format of sparse matrix representation [40]

Hybrid format utilizes ELLPACK and coordinate or CSR formats for representation of partial matrices that exhibit more efficient representation with each formats: ELLPACK is used for partial matrices with uniform number of non-zero rows or columns; coordinate or CSR format is used for non-uniform varying partial matrices.

Packet format utilizes packetizing of matrices (usually representing a mesh), partitioning the original matrix onto several smaller matrices, which are more compact than the original matrix (Figure 3.3-5).

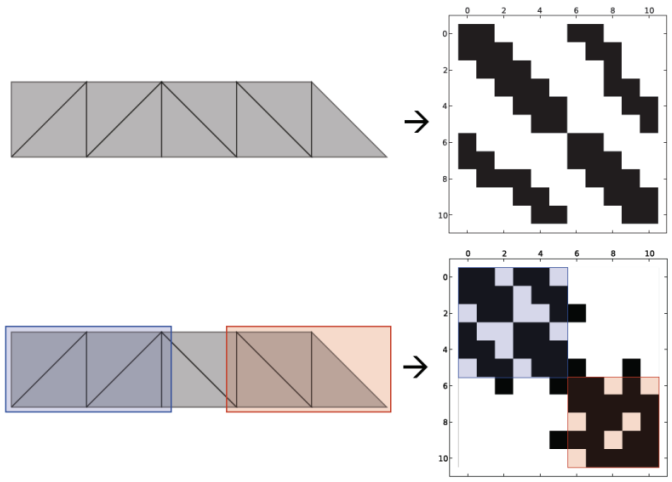


Figure 3.3-5 Packetizing of mesh matrices for packet format of sparse matrix representation [40]

As a result of this operation two small matrices in Figure 3.3-5 can be represented as packet matrices (Figure 3.3-6).

$$\texttt{pkt0} = \begin{bmatrix} (0,0) & (0,1) & (0,3) & (0,4) & * & * & * & * \\ (1,0) & (1,1) & (1,2) & (1,4) & (1,5) & * & * & * \\ (2,1) & (2,2) & (2,5) & (4,0) & (4,1) & (4,3) & (4,4) & (4,5) \\ (3,0) & (3,3) & (3,4) & (5,1) & (5,2) & (5,4) & (5,5) & * \end{bmatrix}$$

$$\texttt{pkt1} = \begin{bmatrix} (0,0) & (0,1) & (0,3) & (4,1) & (4,2) & (4,3) & (4,4) \\ (1,0) & (1,1) & (1,2) & (1,3) & (1,4) & * & * \\ (2,1) & (2,2) & (2,4) & * & * & * & * \\ (3,0) & (3,1) & (3,3) & (3,4) & * & * & * \end{bmatrix}$$

Figure 3.3-6 Packet matrices resulted from packetizing operation [40]

Packet format is suitable for small and patterned SNNs. However, for larger and denser SNNs it is hard to find the optimum packet partitioning. At the same time, if heuristics are applied, it is not an impossible task.

Chapter 4 Implementation Strategies

This chapter reviews and provides examples of major implementation strategies involved in building of the systems of SNNs: integrated circuits, programmable logic and parallel software.

4.1. Integrated circuits

SNNs implemented as integrated circuits using analog VLSI have been defined as neuromorphic engineering by Carver Mead in late 1980s [42]. As a rule, neuromorphic circuits operate in the subthreshold region of MOSFET transistor operation. IC-implemented SNNs usually find applications in prosthetics due to small sizes, as in artificial retina [43], artificial cochlea [44]. However, they also find implementation in system modeling, robotics, image processing and other application domains.

As a rule, the systems implemented in analog VLSI employ array-type architecture, in which each neuron is mapped to an individual circuit cell. For example, in [45] an array of 60x40 IF neurons is implemented with the arbitration circuit responsible for propagation phase (see section 3.1).

Address event representation (AER) protocol is often considered in array-type architectures in order to propagate spike events. In this protocol each neuron has an address. An arbitration circuit resolves generated simultaneous spikes using time multiplexing in synchronous or event-driven fashion depending on the system used. It extracts spike targets stored in memory and delivers synaptic events to them.

Time multiplexing of analog neurons is problematic, because it disrupts continuity of analog VLSI cell's state and requires storing and loading of its model variables for each multiplexed neuron. From perspective of IVP and with analog implementation it requires D-to-A and A-to-D conversion and an additional startup

circuitry to bring the state of the analog cell to its initial value. Thus, the entire array of neurons simulates only a specific part of network where it is mapped to without changing it.

An example of an axon hillock circuit designed by Carver Mead is presented in Figure 4.1-1.

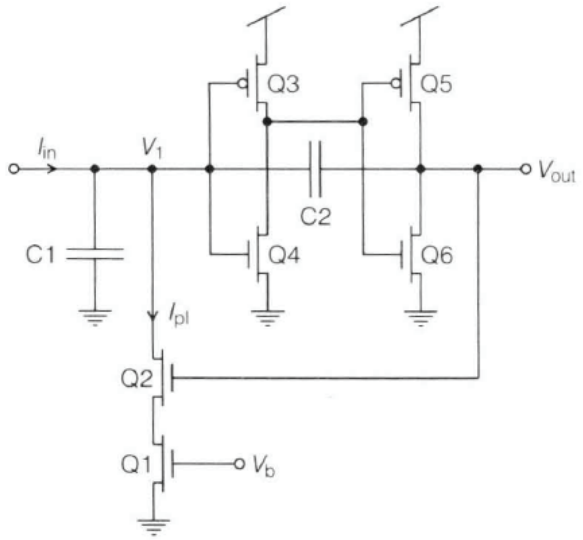


Figure 4.1-1 Axon hillock circuit [42]

In this circuit injected current I_{in} is integrated as a charge on $C_1 + C_2$ since drain of Q_6 is at the ground when node V_1 hasn't yet reached switching threshold voltage of inverter Q_3Q_4. Once this switching threshold is reached, V_1 is inverted twice and as a result, V_{out} rises to V_{DD}. As a consequence, the value of V_1 is set by voltage divider formed with C_1 and C_2. This value relative to switching threshold voltage of the inverter Q_3Q_4 determines a voltage value by which V_1 has to drop in order to terminate the spike. At the same time current I_{pl} controlled by V_b relative to current I_{in} defines the time of discharge at that moment.

In general, this circuit can be viewed as a switched capacitor circuit, in which C_2 plays a dual role: it defines both the time of a spike and the time of charge, which models refractory period.

Although this circuit is simple, it is not optimized for power consumption due to inverter switching times, nor does it provide control over spiking threshold during its operation. An example of low-power integrate and fire neuron is presented in Figure 4.1-2 [46].

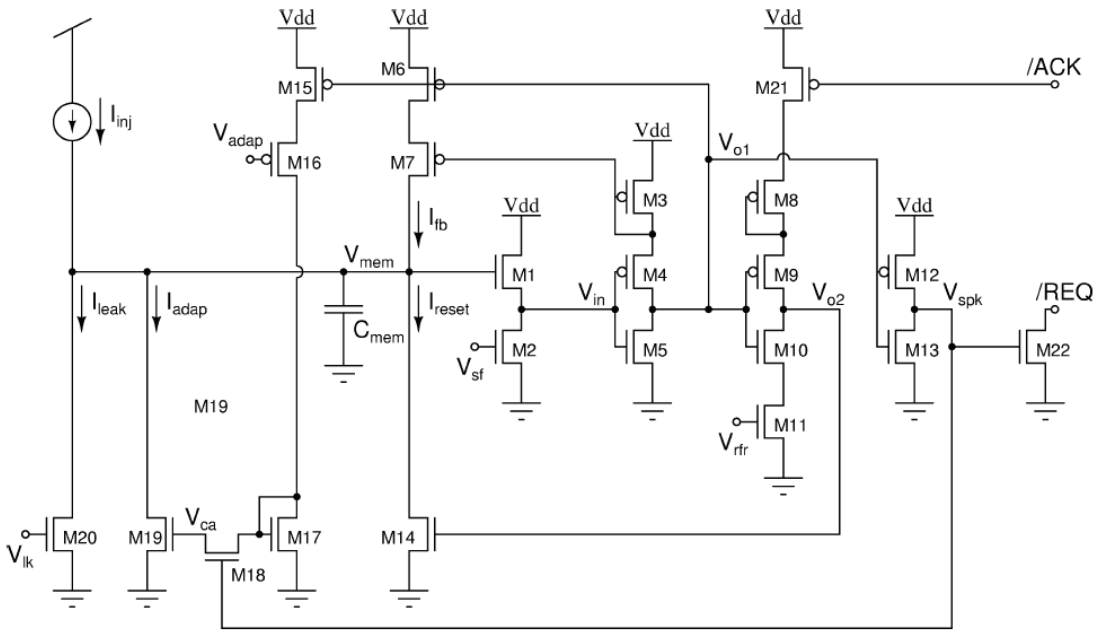


Figure 4.1-2 Integrate and fire neuron [46]

In this circuit injected current I_{in} with subtracted leakage current I_{leak} is integrated as a charge on membrane capacitor C_{mem} rising its potential V_{mem}. This voltage is further buffered and translated to V_{in} with a little loss due to body effect of source follower amplifier M_1M_2. Once V_{in} reaches switching threshold voltage of inverter M_4M_5, its output voltage V_{o1} starts to drop providing positive feedback to the node V_{mem} via M_6 and current mirror M_3M_7. Consequently, feedback current I_{fb} accelerates membrane potential rise at node V_{mem}, which makes switching faster compared to that in

previous circuit, and therefore, draws less current via inverter M_4M_5 and dissipates less power. After inverter switches, signal V_{mem}, doubly inverted via chain of inverters M_4M_5 and $M_{12}M_{14}$, appears as a spike at node V_{spk} and triggers REQ command to the IO interface. Once REQ command is acknowledged with external ACK command M_{21} activates transistor stack $M_8{-}M_{11}$. As a consequence, signal V_{mem}, doubly inverted via another chain of inverters M_4M_5 and M_9M_{10}, appears at node V_{o2} and activates M_{14}, which opens a path for current I_{reset} and resets the membrane potential V_{mem} to the ground. After this happens V_{mem} deactivates inverter chains mentioned above and clears REQ to high impedance. However, V_{o2} is not reset to the ground instantaneously, rather its cause, the charge stored at the gate of M_{14}, leaks through the stack of $M_{10}M_{11}$ during some period of time, a refractory period, controlled by voltage V_{rfr}. During this period it is not possible to generate a spike since I_{reset} is still present.

Once V_{in} reaches switching threshold voltage of inverter M_4M_5, as described above, its output voltage V_{o1} starts to drop activating yet another feedback path via PMOS device M_{15}. The current in the branch $M_{15}{-}M_{17}$ controlled by V_{adap} is integrated as a charge at the gate of M_{19} rising its potential V_{ca} since M_{18} is activated during a spike event. As it is seen from the circuit, voltage V_{ca} defines current I_{adap}. At the same time, the value of V_{ca} is determined by only the spiking rate assuming that other parameters affecting it, such as V_{adap} and leakage currents at the gate of M_{19}, are fixed. As a consequence, the spike signal frequency adapts to the injected current via this negative feedback path.

Mathematically, this circuit can be described by the following current equation:

$$C_{total}\frac{d}{dt}V_{mem} = I_{inj} - I_{leak} + I_{fb} - I_{adap} \qquad (4.1\text{-}1)$$

where C_{total} is total capacitance seen at the input node. Considering that transistors operate in the weak inversion region and after a few transformations, this equation can be modified to (see detailed derivation in [46]):

$$C_{total}\frac{d}{dt}V_{mem} = I_{inj} - I_{leak} + I_1 e^{-\frac{k^2(V_{mem}-V_{sf})}{U_T}} - I_0 e^{-\frac{k^2(\gamma V_{mem}+V_{ca0})}{U_T}}$$
$$\gamma = \frac{C_{gd\ at\ M19}}{C_{gd\ at\ M19} + C_{gs\ at\ M19}} \quad (4.1\text{-}2)$$

where V_{ca0} is steady state voltage at C_a, U_T - thermal voltage, I_0 - dark current through M_{19}, I_1 - constant current through $M_4 M_5$.

As it can be seen, this equation is comparable to the system of equations (2.2-3) except that in this equation an additional exponential current term is present. It models Na^+ionic current that creates action potential upstroke.

4.2. Programmable logic

Most of programmable logic implementations are FPGA-based. As a rule, limiting factors for FPGA-based SNN are the number of multipliers and the data type used (floating point vs. fixed point vs. integer). Consequently, simpler models such as IF are usually implemented, where multiplication is completely omitted [47] or reduced to the power of two, and floating point is replaced with fixed point [48]. Digital logic gates such as “AND” gate can be utilized to model synaptic transmission (Figure 4.2-1).

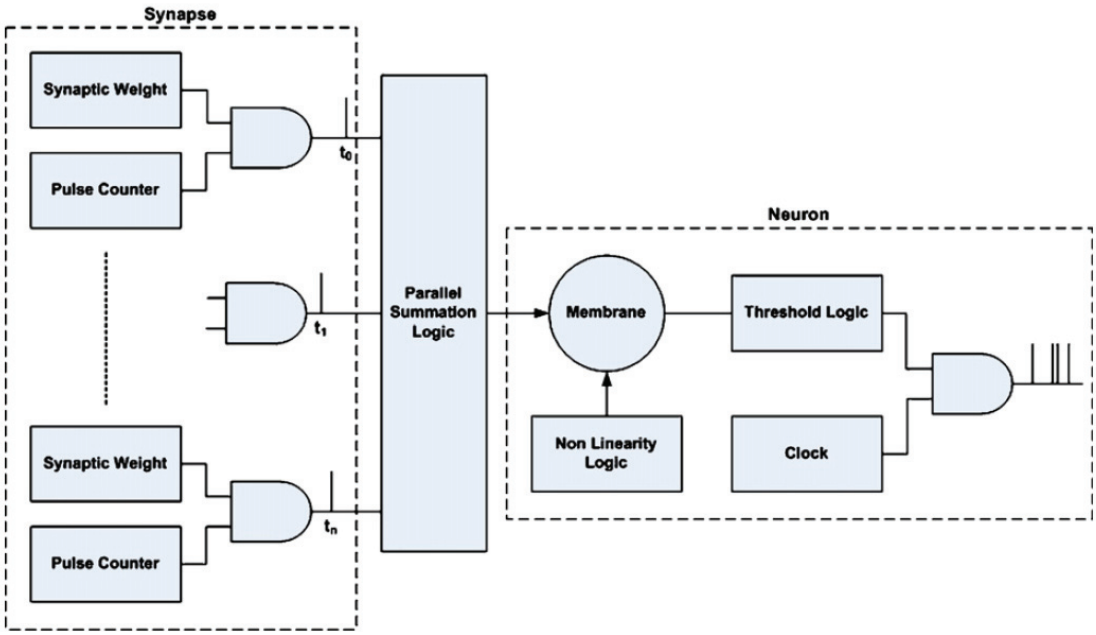


Figure 4.2-1 Synaptic transmission using AND gates [48]

Another approach for reduction of logic utilization and achieving larger network implementation is multiplexing. For example, in [48] soft processors are used for multiplexing neurons computing coordinate transformation. Fixed point and Euler method are applied. The implementation is targeted for large SNNs using STDP synapses and IF neurons. As a result, the system is able to simulate big networks (18,000,000 to 26,986) with large numbers of synaptic connections per neuron (1 to 1000).

FPGAs are suitable for pipelining architectures. An example of such (RT spike) is demonstrated in [49] as software/hardware computing platform with PCI-based communication and FPGA accelerator board (Figure 4.2-2).

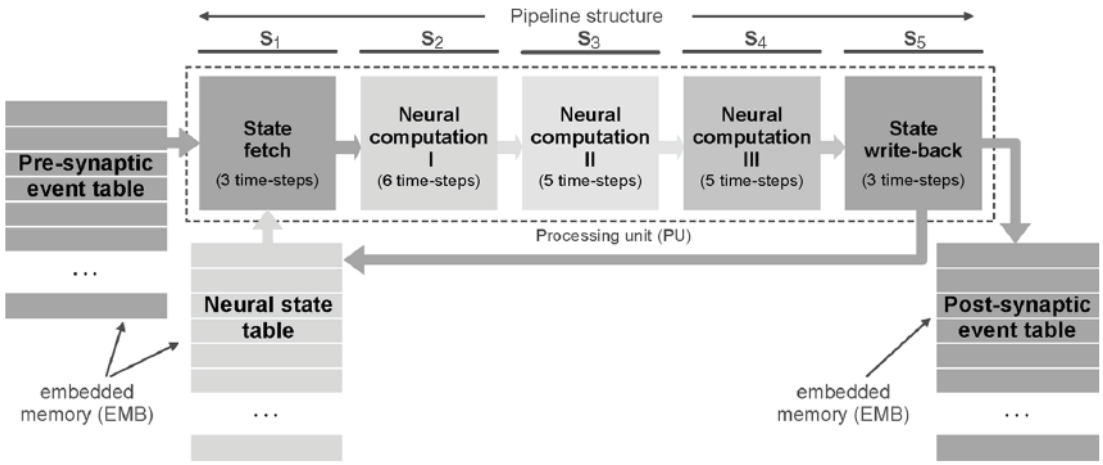


Figure 4.2-2 RT-Spike pipelined architecture [49]

Software is responsible for connectivity and configuration data. PCI bus provides communication of spike events: from software pre- and from FPGA post- synaptic events are sent as time sliced chunks or epochs. FPGA updates state variables through the pipeline.

In [50] a real-time system based on leaky IF model with fixed point arithmetic was implemented for simulation of over 1000 neurons. The system is targeted for applications in robotics and uses SIMD architecture (Figure 4.2-3). An array of neural processing elements (NPEs) process the same instruction issued by the sequencer to all NPEs. Input stimuli enter NPEs via two input modules, and responses leave via two output modules.

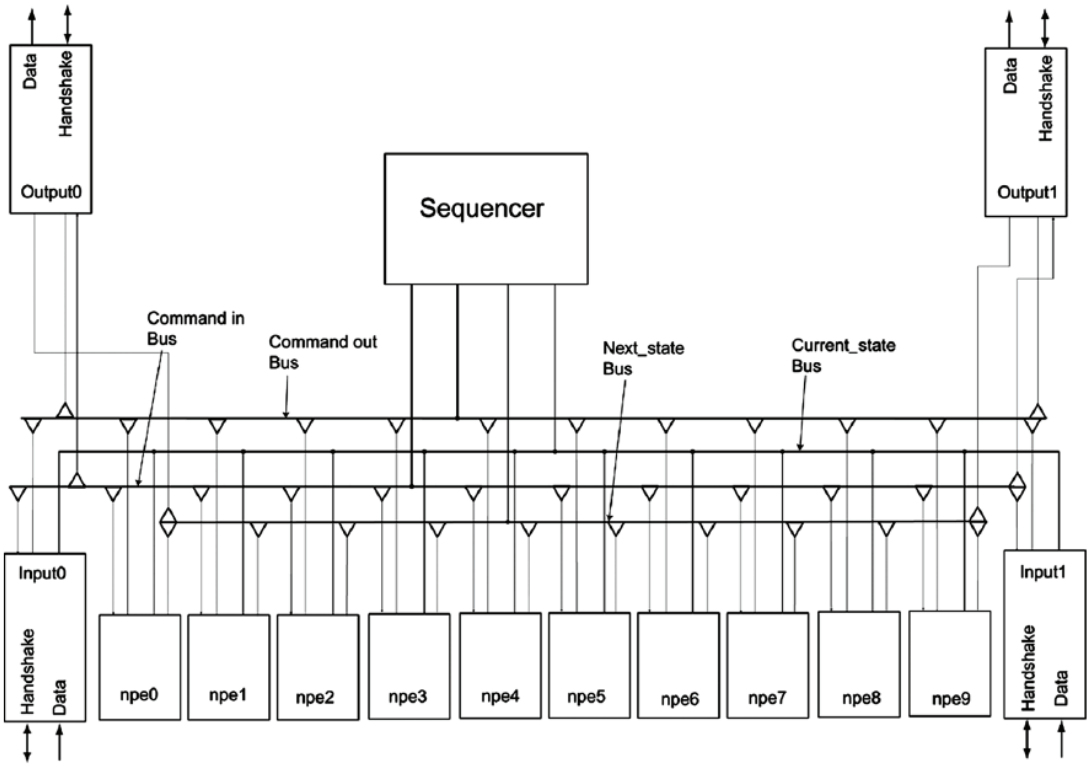


Figure 4.2-3 SIMD architecture of FPGA-based neuroprocessor with IF neurons [50]

An Izhikevich-based SNN has been implemented on FPGA in [51] (Figure 4.2-4). Fixed point arithmetic is used with modified model parameters to fit power of two multiplication and division. Five-stage two-pipeline event-driven architecture provides maximization of time parallelism with 1ms resolution with execution time of 200ns. Precision vs. resource utilization tradeoffs are exploited.

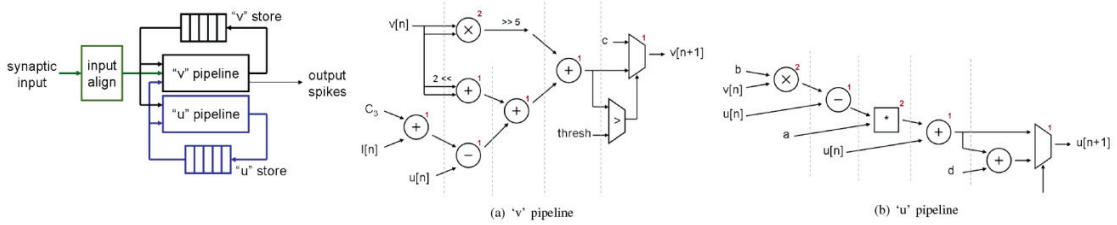


Figure 4.2-4 Pipeline architecture for Izhikevich neuron [51]

In general, although FPGA-based systems are capable of simulating SNNs in real time by means of multiplexing techniques, they limit the data type used in calculations to fixed point arithmetic. Neuron membrane models utilized in FPGA-based systems are usually simple (for example, IF model). Simple numerical integration methods (discussed in section 3.2) are usually used.

4.3. Parallel software systems

According to section 3.1 the propagate phase of SNN simulations requires delivering spike events to their targets with each simulation step, be it an integration step in a synchronous system or an event in an asynchronous system. If implemented in a message passing system, a communication-bounded model may incur an overhead [52]. At the same time, dedicated shred memory systems are costly. Recent trends in GPU computing identify GPU systems as cost effective solutions with high memory bandwidth suited for scientific simulations. Thus, this section concentrates on GPU-based implementations for SNN systems and specifically, on CUDA-enabled GPUs.

4.3.1 Essential Compute Unified Device Architecture (CUDA)

Proprietary to Nvidia Corporation, but freely available, SDK CUDA provides a framework for GPU programming using standard C or C++ library calls. The architecture is specifically designed for supporting the development of general purpose computational tasks on GPUs. CUDA targets scalable programming across CUDA-enabled devices. Since device hardware varies, CUDA assigns its features to every device using a notion of compute capability, which defines the set of features that a device can offer. Compute capability 1.3 and device GTX260 are considered in this discussion. CUDA provides its own compiler, emulation mode, debugging tool and the profiler. CUDA supports

concurrently executed multiple devices, OpenGL and Direct3D interoperability. CUDA can execute on Linux, Windows and Mac platforms.

One of the most important features of CUDA is a support for heterogeneous computing and programming. CUDA-enabled devices communicate with a generic host PC motherboard and host CPU via PCI Express bus. Thus, CUDA allows program execution on the device and on the host and development of the code as a single program, where device part of the code is placed into special functions called kernels (Figure 4.3-1).

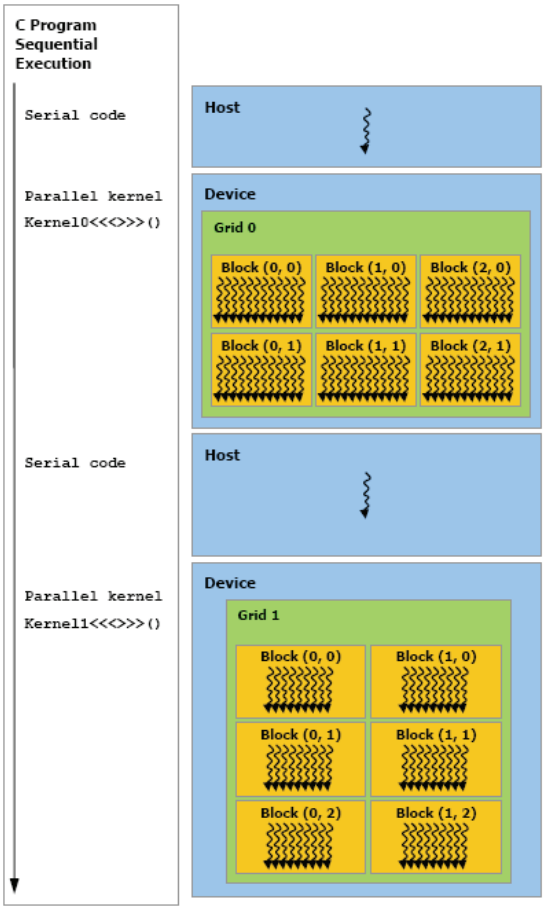


Figure 4.3-1 Heterogeneous programming [53]

The main purpose of a kernel is to divide a computational task and exploit parallel execution as well as to separate device code from host code. The division of a task and making it parallel is based on notions of thread, block and grid. A thread is an independent copy of executing program at runtime with its own state and variable set. A block is a collection of threads. A grid is a collection of blocks. All threads of the grid are scheduled to execute on a device.

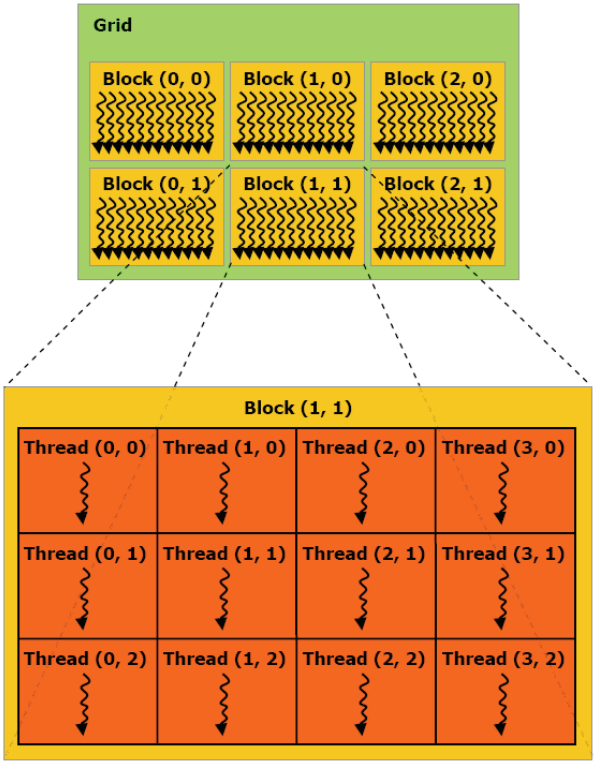


Figure 4.3-2 Thread hierarchy [53]

Threads can be packed into blocks using 1D, 2D or 3D indexing, and blocks can be packed into a grid using 1D or 2D indexing. The resulting indices, called thread ID and block ID respectively, are convenient for data-parallel computations, because each data piece can be allocated to threads/blocks based on these IDs. Granularity of allocation can be as fine as at the bit-level.

Device and host code can be executed asynchronously and concurrently or they can be synchronized. In the former case control is returned to the host thread right after the call to device code, the kernel.

Each device is a collection of streaming multiprocessors (SM). Streaming multiprocessor consists of several scalar processor cores, two special function units for transcendentals, a multithreaded instruction unit and on-chip shared memory (Figure 4.3-3).

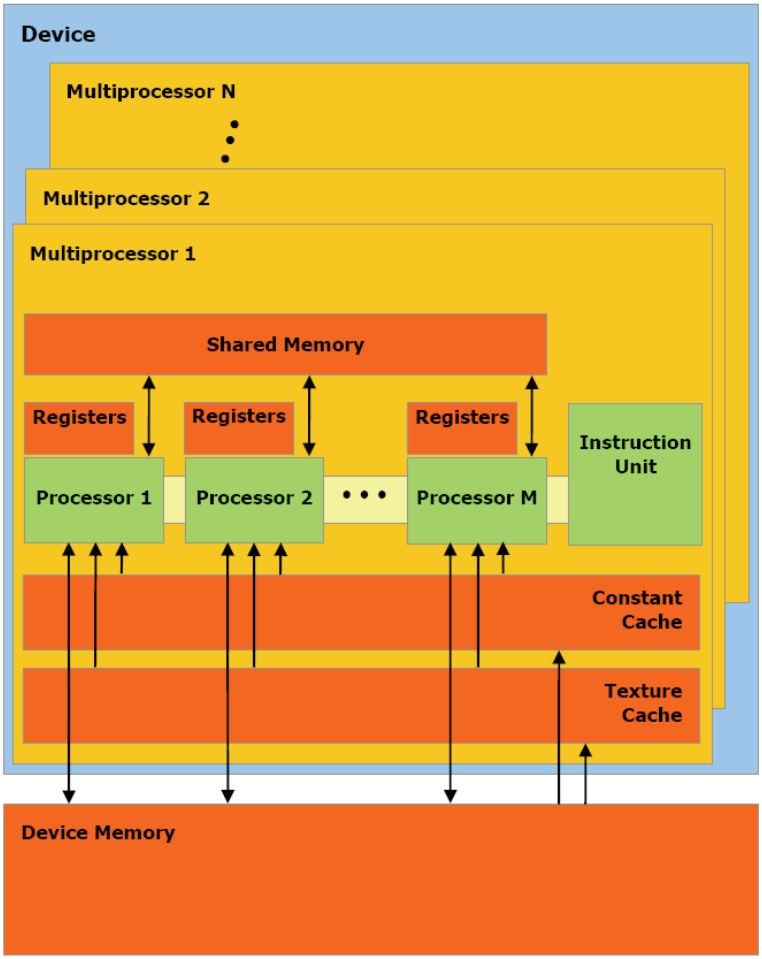


Figure 4.3-3 Hardware architecture of streaming multiprocessor [53]

Device schedules blocks for execution on SMs when the host calls the kernel (Figure 4.3-4). If there are more blocks than SMs then blocks are executed sequentially (in fact, it is sets of active blocks executed sequentially rather than one block, see clarification further). Thus, execution scalability is achieved across devices of various numbers of SMs. In this light blocks are required to execute independently. Each block can be viewed as a unit of scalability. Allocation of blocks to SMs doesn't change within application runtime. Number of SMs on device defines its computational power.

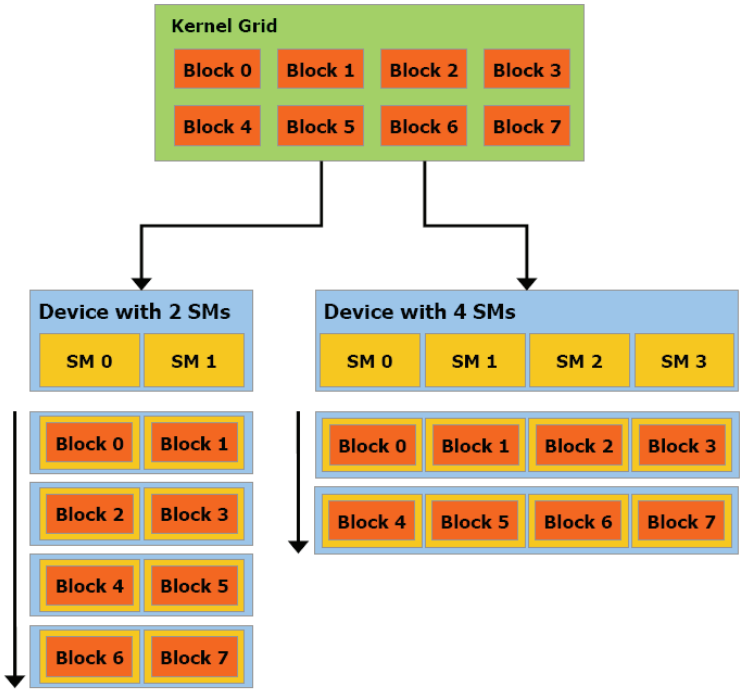


Figure 4.3-4 Allocation of blocks to streaming multiprocessors [53]

Within each block there might be read/write memory transactions. Because certain types of device memory transactions are very costly, they are separated from other transactions and scheduled separately in order to facilitate high efficiency of program execution by overlapping these transactions with other non-memory operations. The dependencies are preserved. However, in order to maximize the throughput, it is essential

to have balance of threads scheduled for memory transactions and threads executing on SMs. Thus, CUDA has notion of maximum possible active threads per SM. For devices with compute capabilities of 1.3 it is 1024 threads. If this number is achieved then the device is operating at full occupancy: overlapping execution of memory transactions with other transactions is hiding memory latency.

However, CUDA has a restriction, maximum allowable number of active blocks. For 1.3 devices it is 8. Thus, a block size has to be at least 128 threads in order not to exceed this number if operating at full capacity of 1024 threads. Within this boundary the allocation of threads per block is program-specific, as long as it tries to maximize the number of threads and approach 1024. However, other restrictions related to SM resource utilization apply (discussed further). Sets of consecutive active blocks are scheduled and executed on each SM sequentially rather than single blocks.

At the same time, although it is said that threads are executed within each block concurrently, they are actually executed in small batches called warps, 32 threads each for 1.3 devices. Each warp is scheduled and executed in sequence on each SM core. Threads are allocated to warps based on thread ID in increasing consecutive order starting from ID zero. SM scheduler issues a single instruction to the entire warp. Threads in a warp are free to take different execution paths (branch divergence). However, these execution paths are processed in sequence until they reach common execution path and continue in parallel from that point. Thus, although CUDA gives this flexibility (naming it SIMT, single instruction - multiple threads) compared to SIMD architectures, it introduces a penalty due to serialization. At the same time, each warp is free to execute a different code without incurring this penalty.

The device has several memory types, which have their own distinct properties (Figure 4.3-5).

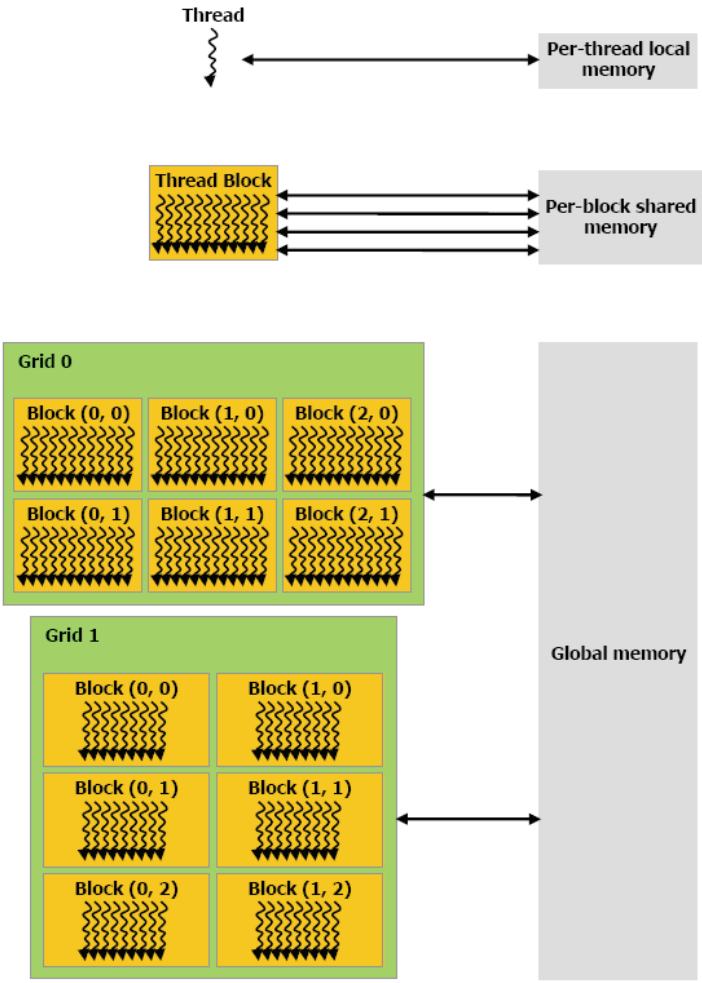


Figure 4.3-5 CUDA device memory hierarchy [53]

Values used in description of memory hierarchy below are related to compute capability 1.3 and GTX260 device.

(1) 32-bit registers is the fastest type of memory, generally with zero access time. There are 16384 registers per SM in GTX260 device. Each thread operates on its own set of registers. More active threads result in fewer registers per thread. For maximum occupancy of 1024 threads, there are only 16 registers per thread. Therefore, if executing algorithm requires more than 16 registers, the number of active threads has to be reduced, which results in non-optimal execution.

(2) Shared memory is on chip L1 cache type read-write memory with a fast access (0.25 CLK cycles) and a small capacity (16 KB per SM). Every thread of a block can access shared memory on SM that this block is executed on, but not the shared memory on any other SM. At the same time, as a resource this type of memory is shared among all active blocks and active threads executing on this SM. There are 8 blocks or less per SM required for maximum occupancy. Thus, there are 2 KB or more that can be allocated per block up to 16 KB in case of a single block. Therefore, executing algorithm has to minimize shared memory utilization per block especially if there are multiple active blocks. In summary, as long as all blocks within a set of 1024 maximum active threads require 16 KB or less of shared memory, the algorithm is not shared memory bounded. If it is not the case, the number of active threads and/or blocks has to be reduced, which results in non-optimal execution. There is another restriction, however: the maximum number of threads per block is 512. As a result, the minimum number of active blocks for maximum occupancy of 1024 threads is 2. Data in shared memory has a lifetime of block execution time.

Shared memory access is done sequentially by half-warps (parallel within each half-warp). The access is done via 16-line (or bank) terminal, where each line provides 32-bit access to all memory locations which fall under: *(shared memory addresses) mod 16 + line number*. As a consequence, if any bank in half-warp access is required more than once, the access through that bank is serialized. For 32-bit data it happens if stride of accessed data structure is even (Figure 4.3-6).

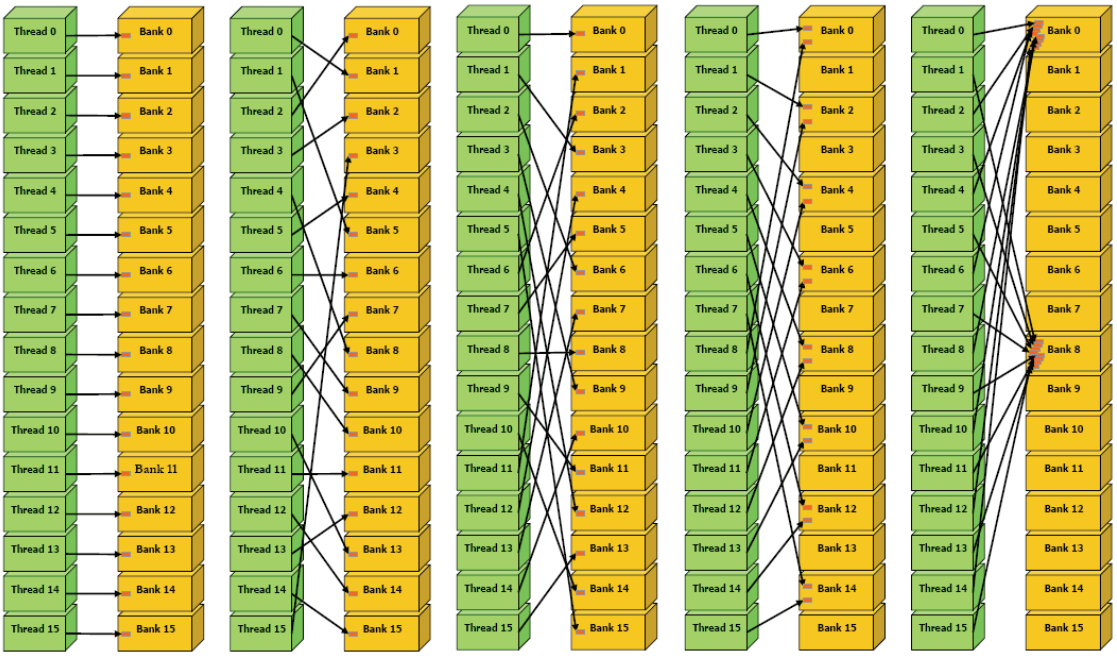


Figure 4.3-6 Shared memory access patterns. 1: linear addressing with a stride of one 32-bit word. 2: random permutation. 3: Linear addressing with a stride of three 32-bit words. 4: Linear addressing with a stride of two 32-bit words causes 2-way bank conflicts. 5: Linear addressing with a stride of eight 32-bit words causes 8-way bank conflicts [53]

If several threads access the same address, the hardware tries to optimize line utilization and broadcasts the result of line access, a broadcast word, to respective threads. However, broadcast access pattern is not always guaranteed to be detected by the hardware (Figure 4.3-7).

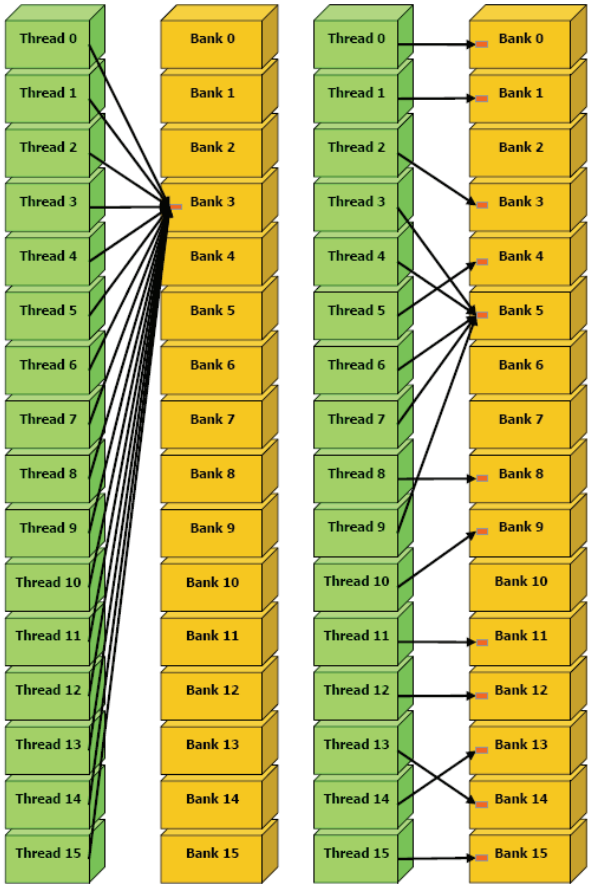


Figure 4.3-7 Broadcast access patterns. Left: This access pattern is conflict-free since all threads read from an address within the same 32-bit word. Right: This access pattern causes either no bank conflicts if the word from bank 5 is chosen as the broadcast word during the first step or 2-way bank conflicts, otherwise.

(3) Local memory is a DRAM type read-write memory with the highest latency (400-600 CLK cycles). It is private to each thread and has a lifetime of the thread. It is utilized only in certain cases determined at compile time: automatic variables that are likely to be placed in local memory, large structures or arrays that would consume too much register space, and arrays for which the compiler cannot determine that they are indexed with constant quantities.

(4) Global memory is a DRAM type read-write memory with the highest latency (400-600 CLK cycles) and the largest capacity (1-2 GB). Every thread can access global

memory. Data in the global memory has a lifetime of application. For optimal global memory access there are two requirements: 1) Data type has to be 8-, 16-, 32-, 64-, 128-bit words or aligned to these sizes; 2) An access for threads belonging to a half-warp has to be aligned to a specific memory grid depending on data type size. Namely, each thread in a half-warp accesses a single piece of data within a data segment of 32 bytes (for 8-bit data), 64 bytes (for 16-bit data), or 128 bytes (for 32-, 64-, 128-bit data). Any combination of accessing addresses of this piece of data within the segment can be done without a loss of efficiency (Figure 4.3-8).

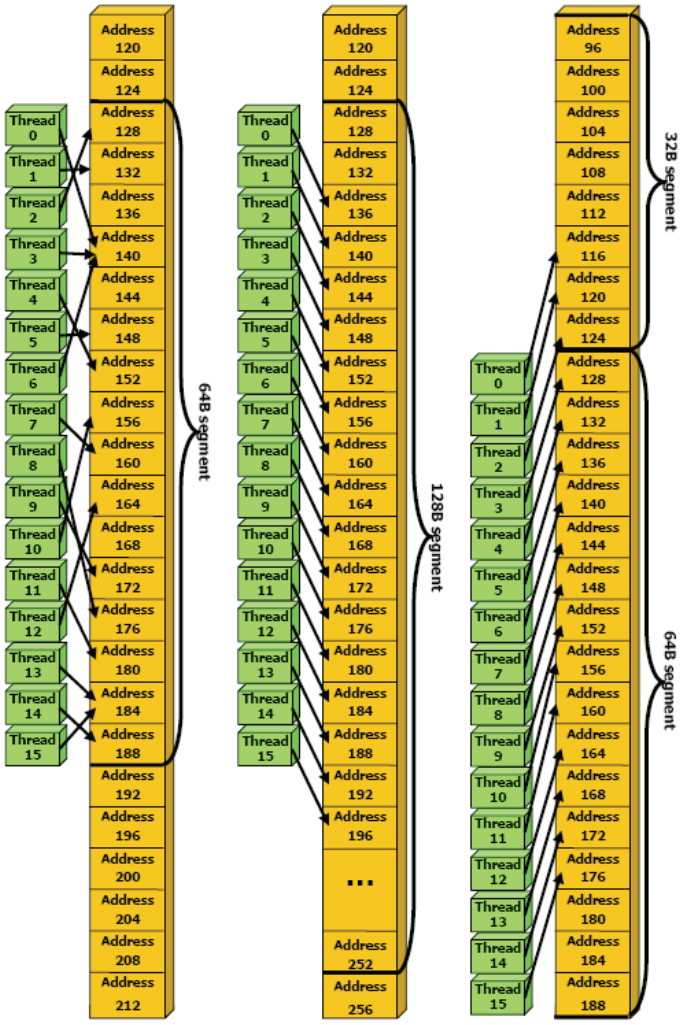


Figure 4.3-8 Examples of memory access. Left: random float memory access within a 64B segment, resulting in one memory transaction. Center: misaligned float memory access, resulting in one transaction. Right: misaligned float memory access, resulting in two transactions

Segmentation is done at allocation time starting from the beginning address of allocated data structure. All beginning addresses are aligned to a grid of at least 256 bytes. 128-bit data require at least 2 memory transactions. The size of transactions is automatically minimized by half if half or more of accessed data is unused. If segmentation condition is not satisfied, then access is serialized linearly with the number of accessed segments.

(5) Constant memory is a DRAM type read only memory with the highest latency and capacity of 64K. Constant memory is cached with 8K of cache per SM for latency reduction purposes. Caching is optimized for broadcast access within a warp. Every thread can access constant memory. Data in constant memory has a lifetime of application.

(6) Texture memory is a DRAM type read-write type of memory with the highest latency (400-600 CLK cycles). It is mapped to the global memory and cached with 6 to 8 KB per SM. Caching is optimized for spatial 2D locality. Thus, the best performance is achieved if warp access data with addresses located closer together. Cache is synchronized with global memory only at kernel launch. Every thread can access texture memory. Data in texture memory has a lifetime of application. Texture memory access provides additional features, such as normalized read mode (value is returned as floating point between zero and one), normalized texture coordinates with interpolation of returned value, and out-of-range coordinate mode (clamp or wrap).

A typical flow of program execution with CUDA is the following: 1) Allocate and initialize memory on the device; 2) Call kernel; 3) Load data from global memory to shared memory and synchronize right after; 4) Process data through the algorithm; 5) Write results back to global memory. 6) Transfer results back to host memory.

Memory transfer between host and device can be done asynchronously or in synchronized manner. Transfer can be performed more efficiently if host memory is declared as page-locked. However, it requires higher host memory utilization. In

addition, page-locked memory can be mapped into device address space if declared as mapped, and can be read by multiple host threads if declared as portable. Besides, computation and memory transfer can be overlapped if computation is divided and executed in streams.

Other convenient operations specific to CUDA are: block level synchronization (warp-level synchronization is intrinsic), atomic operations on integers, warp vote functions to verify if a condition is met for all threads in a warp, memory fence functions that provide function to control scheduling of memory operation.

Although CUDA has IEEE standard support for some floating point operation, in general, the architecture and compiler differences make floating point results received from computation on CUDA devices different from those of host CPU. This difference is insignificant for algorithms with short floating point data dependency path, but results in complete divergence of CPU vs GPU computations for algorithms with continuous iterative dependency path such as in SNNs.

4.3.2 SNN CUDA Implementations

There are only a few implementations of SNN on GPU up-to-date. The most successful ones are presented below.

In [54] a GPU-based SNN with Izhikevich neurons and STDP was reported. Synchronous system is used with Euler numerical integration method with 1ms step (2 x 0.5ms). The speedup of 18 to 25 was achieved for various network sizes and number of synaptic connections (100K – 220K neurons and 100-500 post synaptic connections). The simulation is 1.5 times slower than real time and is bounded by GPU memory bandwidth. A comparison was made between GTX 280 device and Intel Core 2 6400, 2.13 GHz. As expected, GPU implementation functionally diverges from that of CPU due to differences in floating point hardware. The system uses multi-kernel implementation (Figure 4.3-9) in synchronized with CPU kernel launch mode with block size of 60-128 threads.

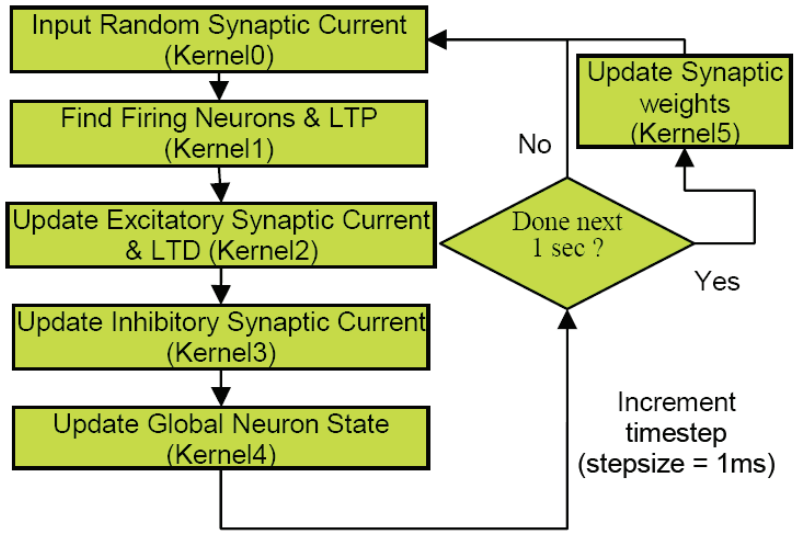


Figure 4.3-9 Flowchart of simulation [54]

Implementation exhibits good scalability and is targeted for large network simulations with a large number of synaptic connections. Simulation accuracy plays a secondary role. As a consequence, the 1st order Euler integration technique is applied. As it is known, Euler method is prone to be numerically unstable with slow converging error, and as a rule, is not often used in scientific computations. However, it depends on application and its requirements to the error tolerance. The choice of synchronous system allows exploiting synaptic parallelism in this case because spike events are tight to the time grid and can be processed in parallel. At the same time, a clock-driven system introduces quantization error due to the time grid. Warp divergence is minimized using buffering technique: tasks are serialized; however, each task is computed using parallel domain decomposition. Thus, thread utilization for computation of each task is maximized.

In [55] both synchronous and asynchronous systems based on leaky IF neuron model computing spike-based convolution for visual pattern recognition have been reported (Figure 4.3-10).

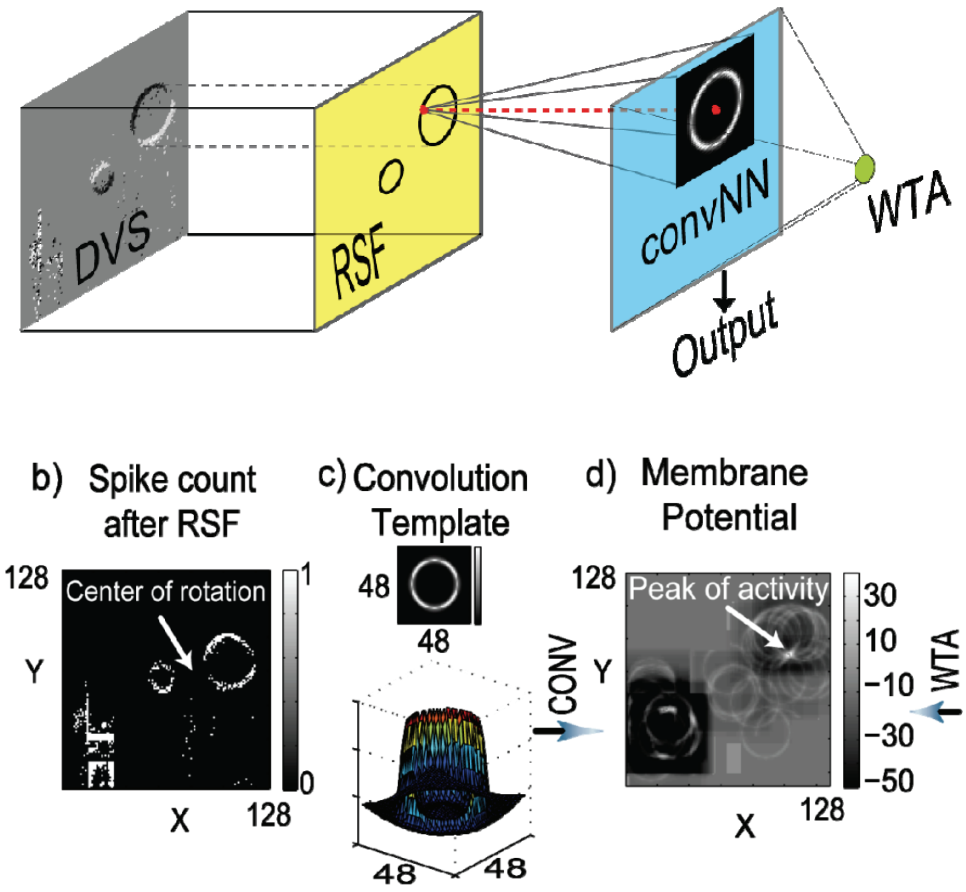


Figure 4.3-10 SNN architecture [55].

The system consists of a dynamic vision sensor (DVS), and a receptor layer that generates asynchronous spike events in response to the projected scene. These events are filtered by refractory spike filter (RSF) for reduction of temporal redundancy: since neurons are not sensitive to synaptic events during refractory period, several consecutive spikes within this period are reduced to one spike. Both DVS and RSF are executed on CPU thread. Spike events are further propagated to the convolution network executed on GPU. In this network each incoming spike event is translated to synaptic events with weights and topology defined by convolution kernel template stored as a texture, which describes desirable feature for recognition. As a result of movement, a spiking map is created for the entire network with a spiking activity highest at the center of the moving

object to be recognized. Post-processing using winner-takes-all (WTA) network leaves only one neuron with the highest spiking rate by suppressing all other neurons. With this architecture, it is possible to track multiple objects since receptor layer can be mapped to multiple convolution layers, each of which is responsible for tracking a specific object with specific features.

Because all incoming spikes are external and there is no need to propagate spikes within the layer (WTA is a separate layer) update and propagation phase are combined. Each neuron is mapped to a single thread. The effect of time step size is exploited: for the asynchronous system the kernel speedup is 1.9 to 6.4 (template size varies from 48x48 to 112x112 neurons). The primary overheads are kernel launch and global memory access (shared memory is not utilized). For the synchronous system the speedup is 3 to 20 while increasing both size of the template and time step (0.1 to 5ms). Quantization error, although significant, does not degrade the quality of detection since learning is not involved and features are simple. Increasing the number of convolution networks to 5 results in top speedup of 35x. A comparison was made between GTX 280 device and Intel Core 2 6400, 2.13 GHz.

One of the current industry and science research leaders and pioneers in the development and deployment of SNN on GPU is Evolved Machines [56]. The company focuses its research on SNNs synthesis based on the synaptic plasticity modeling on GPUs. The primary focus is application of SNNs synthesized according to olfactory cortex and visual cortex for pattern recognition [57]. Developed applications are proprietary to the company.

Chapter 5 Design and Implementation

In this chapter design and implementation of Izhikevich SNN on GPU using PS numerical integration method is presented. The system is composed of several components: neurons, communication framework and interface. Flexibility in parameter variations allows designing application-specific SNNs functioning in real time.

Izhikevich model (see section 2.2) provides an advantage in terms of computational simplicity and biological plausibility at the expense of reduction and modification of the system parametric space, which makes the model not biologically intuitive, but still functionally adequate compared to classic biologically plausible models such as HH model. Thus, IZ model has an advantage weighed against both HH and IF models described in section 2.2.

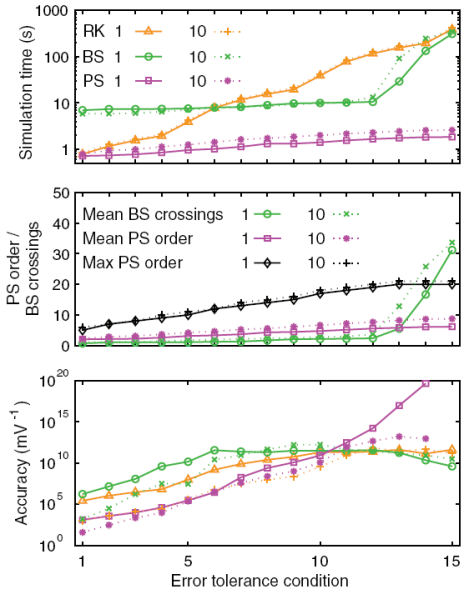


Figure 4.3-1 Izhikevich model current injection results. (Top) Mean simulation time for 1 s simulations with varying error tolerance conditions. (Middle) Adaptive processing statistics. Plots show the mean (over a simulation) number of crossings used by the BS method per step, and the mean and maximum order of the PS method. (Bottom) Simulation accuracy taken as the reciprocal of absolute voltage divergence between test and reference traces. Line styles as in top panel. PS – Parker-Sochacki method, BS – Bulirsch-Stoer method, RK – Runge-Kutta 4th order method. Error tolerance 1E-(condition+1). Time step 0.25ms [2]

PS integration method (see section 3.2) applied to IVP allows implementing asynchronous or hybrid system of SNN (see section 3.1) potentially with any accuracy, limited only to that of data type chosen. In fact, the sequential version of IZ neuron-based SNN with utilization of PS method has been demonstrated achieving full double precision accuracy [2]. Thus, for systems targeting implementations with STDP (see section 2.1.3), this method provides better results than other most commonly used numerical integration methods (Figure 4.3-1). At the same time, considering natural adaptation capability of PS method, namely, its ability to vary the series order (number of summands) depending on Lipschitz constant, which defines local continuity of the solution function, PS method introduces difficulties for GPU-based implementation, because the latter is more suitable for well-structured non-adaptive, non-warp-divergent types of computation (see section 4.3).

IZ model has only a single quadratic term, which limits the number of Cauchy product to only one and reflects in lower GPU resource utilization, compared to HH model, which contains 4^{th} power terms in its membrane equation and most likely won't be able to fit in GPU resource boundaries considering that it requires 19 Cauchy products [2].

Following hybrid system formalism defined in Chapter 3 the system can be described by the overall diagram depicted in Figure 4.3-2, which includes implementation-specific steps.

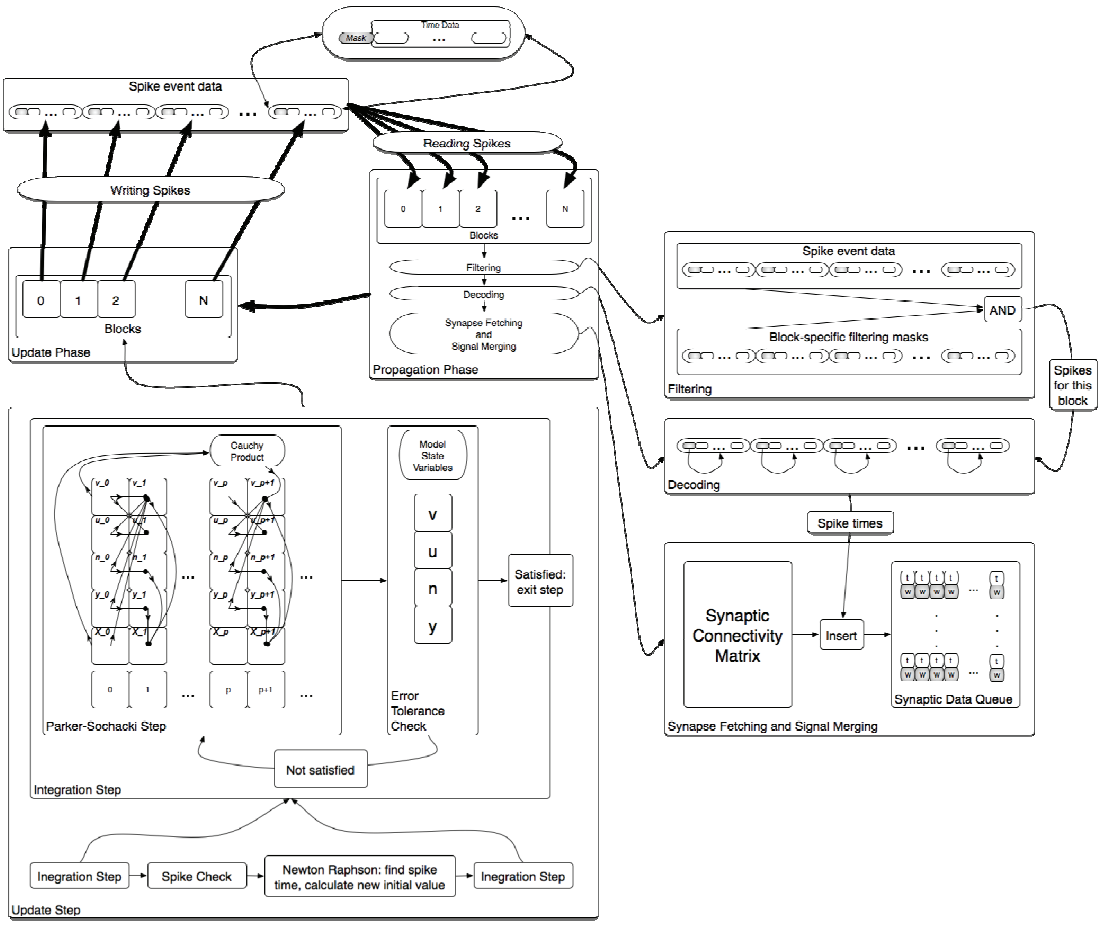


Figure 4.3-2 Block diagram of SNN implementation as a hybrid system

Each of the computational phases with specifics of their implementation is reviewed below.

5.1. Update phase

Izhikevich model with conductance-based aggregated (reduced to a single) synapses can be described by the following set of equations ($v_{rest} = 0$):

$$C\frac{dv}{dt} = kv(v - v_{thresh}) - u - \eta\left(v - E_\eta\right) - \gamma\left(v - E_\gamma\right) + I$$
$$\frac{du}{dt} = a(bv - u),\ \frac{d\eta}{dt} = -\lambda_\eta\eta,\ \frac{d\gamma}{dt} = -\lambda_\gamma\gamma \qquad (5.1\text{-}1)$$

The same model can be described by the following set of equations expressing a single PS step [2]:

$$y(t + \Delta t) = y(t) + \sum_{p=0}^{n} y_p \times (\Delta t)^p, \qquad y = \{v, u, \eta, \gamma\}$$
$$v_1 = \left((\chi v)_0 + E_\eta\eta_0 + E_\gamma\gamma_0 - u_0 + I\right)\frac{1}{C}$$
$$v_{p+1} = \left((\chi v)_p + E_\eta\eta_0 + E_\gamma\gamma_p - u_p\right)\frac{1}{C(p+1)}$$
$$u_{p+1} = a\left(bv_p - u_p\right)\frac{1}{(p+1)}$$
$$\eta_{p+1} = -\lambda_\eta\eta\frac{1}{(p+1)} \qquad (5.1\text{-}2)$$
$$\gamma_{p+1} = -\lambda_\gamma\gamma\frac{1}{(p+1)}$$
$$(\chi v)_p = \sum_{j=0}^{p} \chi_j v_{p-j}$$
$$\chi_j = kv_j - \eta_j - \gamma_j, \qquad \chi_0 = kv_0 - \eta_0 - \gamma_0 - kv_{thresh}$$
$$if\ v \geq v_{peak}: \begin{cases} v = c \\ u = u + d \end{cases}$$

where v is membrane potential, v_{rest} is resting membrane potential (normalized to zero in this model), v_{thresh} is a threshold potential, v_{peak} – action potential escape limiting value, u - recovery variable that models Na^+ and K^+ currents, η - excitatory conductance, γ – inhibitory conductance, Δt - time step, n – maximum PS order, p - current PS order, E_η - excitatory synaptic receptor reversal potential, E_γ – inhibitory

synaptic reversal potential, I – artificially injected current, C - membrane conductance, a is a parameter that describes the time scale of recovery variable u, b is a parameter that describes the sensitivity of u to subthreshold fluctuations of membrane potential, k describes sensitivity of v to the fluctuations of itself, d is a parameter that describes after-spike reset of u, c is a parameter that describes after-spike reset value due to hyperpolarizing outward K^+ current and typically is set to a value less than v_{rest}, $(\chi v)_p$ - second order term that requires an additional Cauchy product, λ_η and λ_γ - excitatory and inhibitory decay rate constants respectively.

This set of equations can be represented graphically as a dependency graph (Figure 5.1-1).

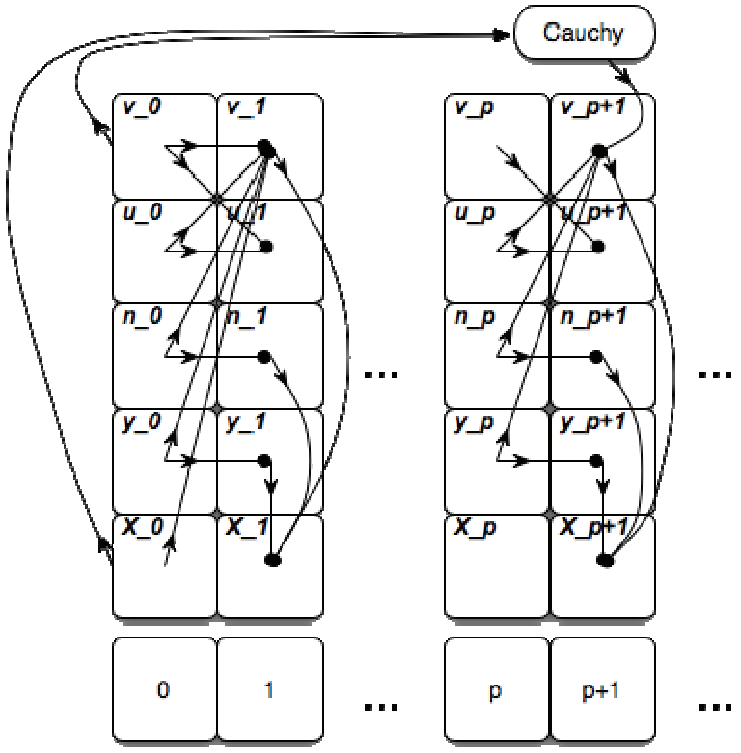


Figure 5.1-1 Dependency graph for PS step of Izhikevich neuron model.

As it can be seen from equation (5.1-2) and Figure 5.1-1, PS step has both data and task parallel operations. Data operations include load-store, error tolerance test,

increment and Cauchy product operations. Task parallel operations include: an update of synaptic conductances, which are loop-level parallel, and partial computations of model variables. However, in general, PS stepping is a sequential process.

Overall diagram of update phase is depicted in Figure 5.1-2

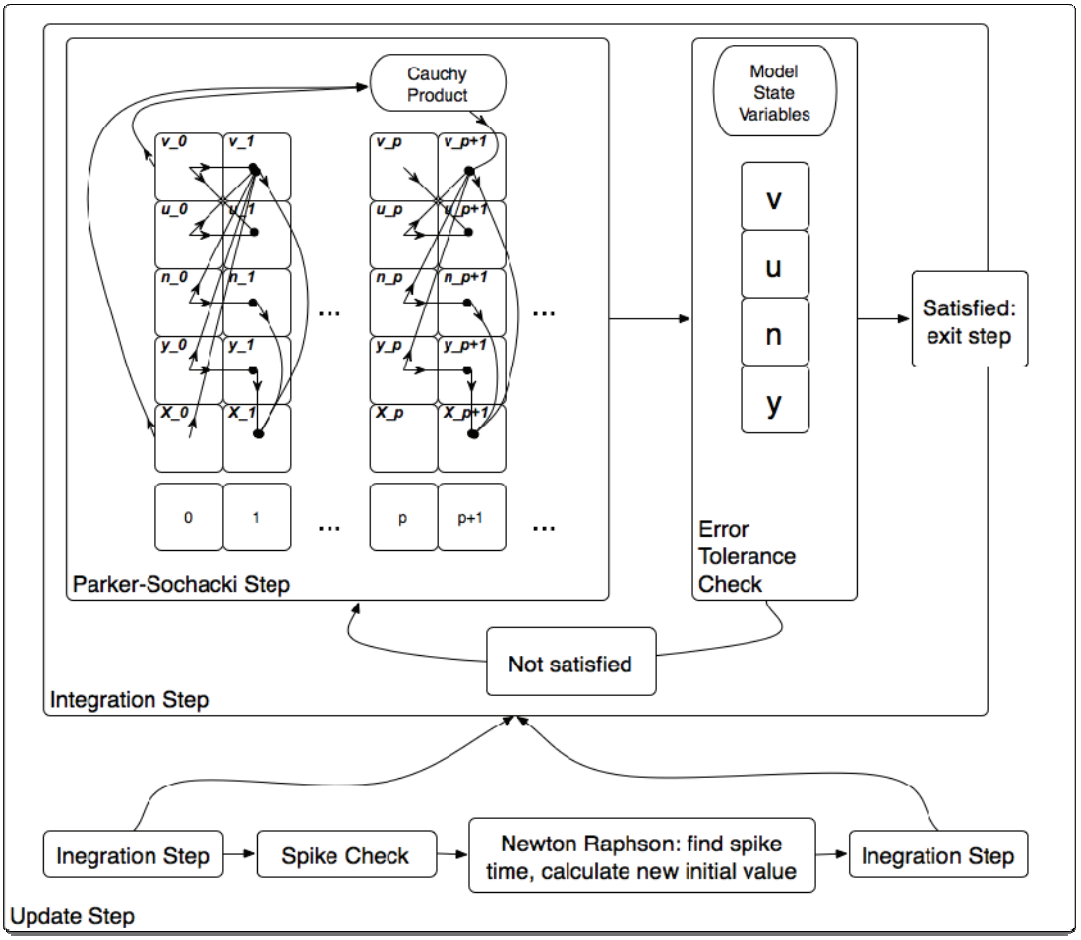


Figure 5.1-2 An integration step in update phase of implemented SNN

The algorithm proceeds with calculation to an order that satisfies given error tolerance conditions. The effective order varies and depends on local Lipschitz constant, which defines smoothness condition: a large value of the constant signifies faster escape of the function toward the infinity (spike), which results in a higher order. The error tolerance verification is done at every step.

For calculating exact spike times Newton-Raphson (NR) method is used for root finding [2]. In this method the root of a polynomial can be found with targeted precision by successive iterations with at least quadratic conversion. Given that function v is monotonically increasing on interval Δt and knowing the fact that at the end of this interval (but not at the beginning) v is more than v_{peak}, it is possible to find the time when v crossed v_{peak} by making this time a root of polynomial function v. This is done simply by down-shifting v by v_{peak}. Applying the tangent equation to the point in time t_n at the end of the time step:

$$\frac{dv(t_n)}{dt} = \frac{rise}{run} = \frac{\Delta v}{\Delta t} = \frac{v(t_n) - 0}{t_n - t_{n+1}} \tag{5.1-3}$$

and rearranging the terms results in NR equation for the next best approximation of the root we get:

$$t_{n+1} = t_n - v(t_n) \Big/ \left(\frac{dv(t_n)}{dt}\right) \tag{5.1-4}$$

Successive iterations of equation (5.1-4) produce a root with targeted precision that corresponds to the number of iterations (Figure 5.1-3).

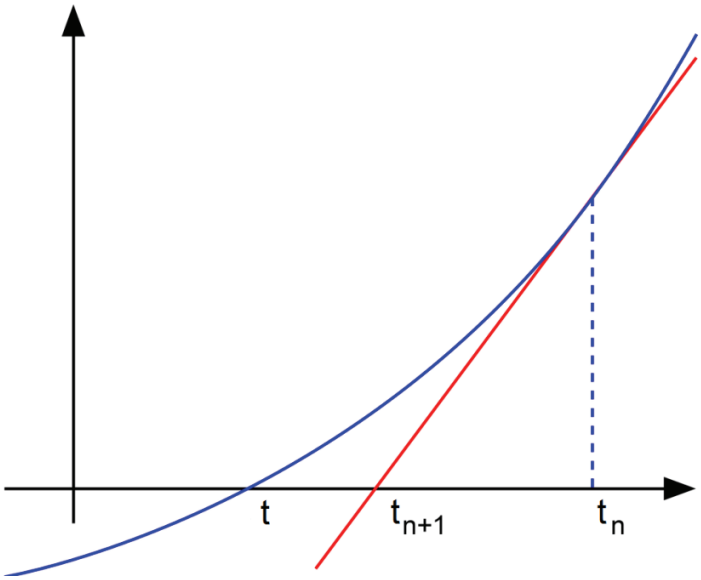


Figure 5.1-3 Newton-Raphson iteration

Consequently, if a spike occurs within an integration step, the values of model variables at the end of the step have to be re-calculated according to the new initial values at the reset time obtained within the integration step. The reset time corresponds to the time obtained via NR method.

According to CUDA model, the efficiency of a computation is maximized if the full capacity of active threads (alternatively active warps) per streaming multiprocessor (SM) is exploited, because memory and register operations processing are overlapped for the entire set of active threads. Thus, the bigger this set the more memory transactions can be hidden. Allocation of threads per block ideally is defined by the level of parallelism per block, but some restrictions hold. First of all, the number of active blocks per SM is limited. Thus, if a thread quantity per block is too small, the capacity is not fully exploited (block bounded). Secondly, in order to utilize the full capacity all active threads have to limit SM register utilization to that of maximum available per active thread. In case of 1024 maximum active threads and 16384 registers per SM, there is only 16 registers per active thread regardless of thread count per block. Yet another restriction is the maximum size of shared memory per SM: if a computation can accommodate the entire set of shared memory available on SM, but use only a handful of threads, it is

memory bounded and allocation of additional active threads would not make sense. Granularity of resource allocation, such as register allocation unit size, warp allocation granularity etc, plays an important role as well. In any case, the block processing by SM is done sequentially in terms of an active set with a size bounded by available resources: SM always forms an active set and processes blocks sequentially by chunks with the size of an active set.

Considering this, the degree of parallel computation can be exploited gradually, with estimation of potential resource limitations and attempts avoiding them. Following hybrid system formalism, defined in Chapter 3, the system performs update and propagation phase. Within the update phase, computation of model variables per each neuron is parallel. However, there is a possibility to exploit additional levels of the parallelism described below.

One of the alternatives is to allocate the core computation to one warp and exploit task parallel operations according to the program flow by allocating them to other warps: one task per warp. In this case, warps responsible for various tasks are only utilized when invoked by the program. The depth of warps is defined by a maximum simultaneous number of parallel tasks. Although this representation results in under-utilized warps, it might provide an increase in efficiency due to parallel task allocation. At the same time, block-level synchronization is required among all warps at each point of a task completion. Considering adaptive nature of the algorithm, which results in warp divergence, total execution time of each block is maximized by the number of divergent spike events within each block. This implies serialization if spikes occur at distinct synaptic events in their sequence for each of the neurons in the block. Besides, with adaptive order processing execution time of iteration has to be at least that of a neuron with maximum PS order. Thus, block-level synchronization demanded by parallel task exploitation results in serialization of the algorithm. Another source of considerable impact is the overhead introduced by program flow control statements required for allocating parallel tasks to specific warps.

A different alternative, which may result in a better scalability, is to distribute computation among all warps sparsely: several neurons per warp. In this case the number

of neurons per warp is defined by resource boundaries. If a computation is such that it allows for extra threads per neuron due to non-optimal resource utilization defined by nature of computation, then these threads can be utilized for data parallel operations. For example, if in Figure 5.1-4 shaded threads in a warp perform the main computation, the other threads participate in data parallel operations whenever there is a need.

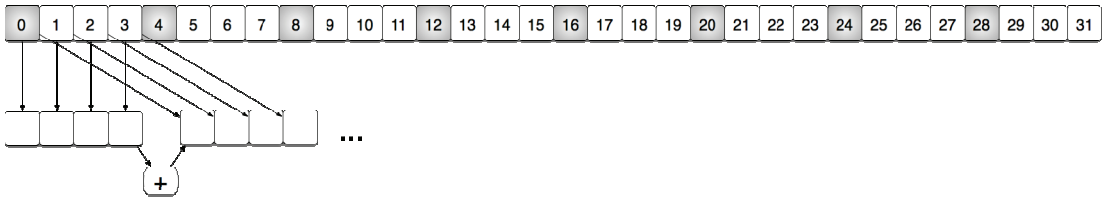


Figure 5.1-4 Partial warp data parallel operations.

Underutilization of these threads is evident due to the fact of sparse data parallel operations including Cauchy product operation on data arrays. An attempt to exploit these operations resulted in increase of execution time compared to that of a simple sparse allocation with unutilized threads. The primary reason for this is believed to be control statements, the result of data parallel size variability for different operations, and due to the adaptive nature of the algorithm. GPU utilization is bounded by the number of registers per thread in the first place and shared memory per block in the second.

Yet another area for exploitation of parallelism in PS step is loop level parallelism (LLP), which is exposed in main PS iteration step. Namely, GABA and AMPA (see section 2.1.2) conductance coefficients can be pre-calculated for computation with dependence on the initial value only. As a result, GABA and AMPA values, as well as some parts of membrane potential computation that only depend on these values in each iteration, can be computed in parallel for all iterations of PS step if maximum order is know *a priori*. An attempt to exploit LLP resulted in an increased execution time. The primary reason for this is believed to be load operations for a coefficient extraction since the maximum order is assumed. In addition, floating point value synthesis operations provided increase in execution time: significant and exponent are stored separately in order to keep precision at the original level.

Based on these results, PS step is kept in a sequential form, leaving parallel exploitation at the level of neurons. Minor optimizations for reduction of register and shared memory utilization have been made. With these optimizations, it is possible to allocate 4 neurons per warp, which results in 32 neurons per block. The algorithm is both register bounded and shared memory bounded: 49 registers per thread and 16 KB of shared memory per block with a single active block per SM, which accounts for 25% of maximum active set utilization.

At the same time, a possibility of another optimization exists. Since the algorithm is adaptive-based, that is the number of PS steps for each neuron is determined at run time and varies, resulting warp divergence is inescapable if more than one neuron is allocated per warp. Besides, NR condition, although much more rare, takes place and diverges a warp. In order to reduce warp divergence and at the same time utilize dynamically allocated shared memory more efficiently, it may be possible to apply warp pipelining technique. In this case instead of allocating each warp to a parallel task, it is allocated to each sequential task and task queues are inserted between pipeline stages.

As a result each thread in a warp executes tasks dynamically as they become available. However, this alternative may incur atomic operation overheads necessary for dynamic queue synchronization.

Another option is to eliminate adaptive order and execute PS step to the maximum order for all neurons and all times. In this case warp divergence can be avoided. However, in any case, grouping neurons with the same number of synaptic events to be executed at a current step can be beneficial.

5.2. Propagation phase

Propagation phase can be either developed as a separate kernel specifically dedicated for translating spike events to synaptic events or it can be incorporated in the kernel that computes update phase. In the current implementation a single-kernel

approach is taken; however, performance results showed that this implementation is not an optimal one.

Communication of spike data between the blocks proceeds in the following fashion: at the end of the update phase all neurons write their spike times to global memory; at the beginning of propagation phase, which follows the update phase, each block reads the entire spike data set, and then translates it to local synaptic events (Figure 5.2-1).

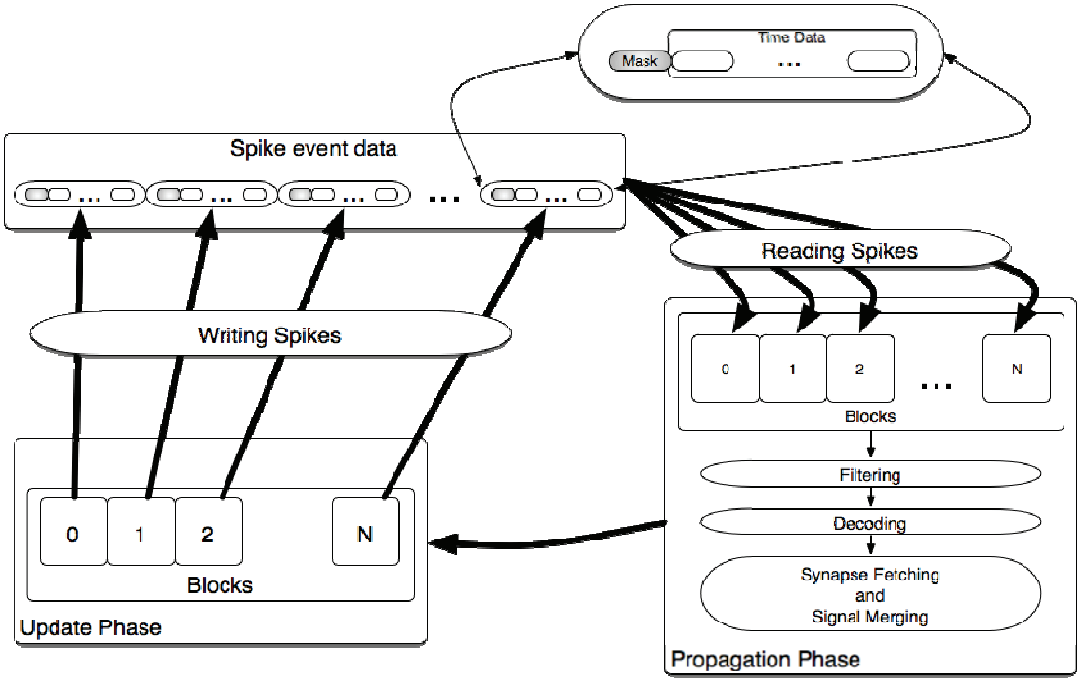


Figure 5.2-1 Data exchange between update and propagation phases

This allows each block to process synaptic data independently. Optimized neuron-to-block mapping targeted for a reduction of the inter-block synaptic connection could be applied in order to prevent each block from reading the entire spike data set or minimize it. Further optimizations of this design are possible in dual kernel implementation.

Alternative implementation of this phase is the case when each block works on a part of a spike data set. In this case there is no need for each block to read the entire spike data set. However, distributing spike events to synaptic queues has to be synchronized

between the blocks since spike data has multiple target blocks and cross-access is inescapable.

Spike event communication requires global synchronization between all blocks after each integration step. This entails a kernel launch with each integration step demanded by CUDA architecture, which does not support global synchronization between blocks during kernel execution. The probability that all neurons in a block produce spikes is very low and depends on the factors that influence a spiking rate: neuron parameters, number of synaptic connections per neuron, the value of injected current, and other factors. Thus, the data size of spike events per block can be tuned to specific application if the maximum number of spikes per block per integration step is known. This maximum is usually less than the number of neurons per block.

Therefore, there is an option to reduce spiking data size based on either global maximum or block-specific maximum. However, it incurs certain computational cost related to data encoding and decoding. Alternatively, there is an option of one-to-one mapping of the spiking data size to that of the number of neurons per block, which incurs the cost associated with more global memory operations.

In this implementation (Figure 5.2-1) each block encodes spiking neuron number using the spike mask (32-bit number). Spike times follow the spike mask in memory layout. The size of the memory allocated to spike times is bounded by the global maximum number of spikes per block per integration step. The framework supports statistical tracking of this maximum as well as the error code that is set when the maximum is exceeded.

After each block receives incoming spike events several stages are performed that provide translation of spike events into target synaptic events: filtering, decoding, synapse fetching, signal merging (Figure 5.2-2).

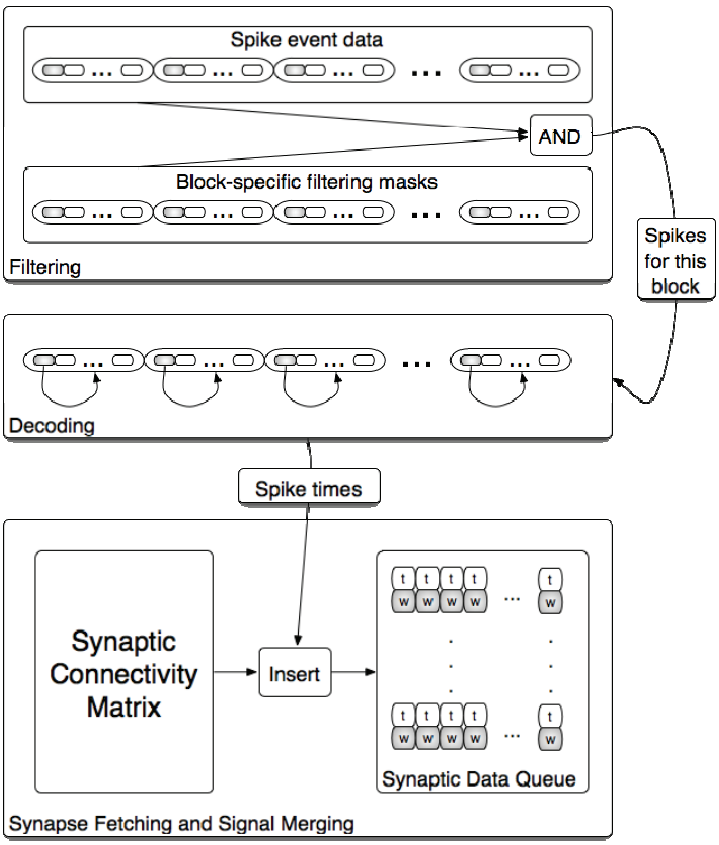


Figure 5.2-2 Propagation phase stages

Filtering of incoming spike events follows network topology and prevents spikes from topologically unconnected neurons being merged in target neuron signal queues. Filtering is done by means of application of bit-wise masks with "AND" operation, which are unique to each block.

Decoding neuron indices of incoming spikes is necessary for relating spike times to their sources. Decoding is currently implemented with a bit-shift algorithm. However, parallel reduction implementation may provide better performance.

After spike times and source indices relation is decoded, the target synapse data is fetched. The efficiency of synaptic data fetch, representation and storage are determined by SNN topology and synaptic data access pattern.

According to [40], ELLPACK format provides a higher bandwidth for structured matrices and hybrid format provides a higher bandwidth for unstructured matrices.

Although structured matrices are not necessarily the case for all synaptic matrices, the modified ELLPACK format is utilized in this implementation with the potential to extend it to hybrid format (ELLPACK-coordinate or ELLPACK-CSR). Indeed, ELLPACK format implies more efficiency for sparse matrices with a uniform number of synaptic connections per neuron. At the same time the part of the program responsible for the synaptic data representation can be modified and tailored to a specific network topology.

Since static domain decomposition has been applied to the network and in order to facilitate 2D access locality of synaptic data, it seems more intuitive to partition the synaptic matrix and obtain smaller matrices each of which correspond to accessing block. At the same time the access of the synaptic data by each block, although constrained to 2D segment, is rather random and depends on spike event sources. Because spiking neurons are most likely to spike again, the access of synaptic data exhibits temporal locality as well. 2D spatial locality and random access pattern identify accessing global memory via texture cache as a suitable memory selection among the choices offered by CUDA. However, temporal cache locality is not mentioned in NVIDIA CUDA Programming Guide [53]. Figure 5.2-3 displays the representation of synaptic matrix.

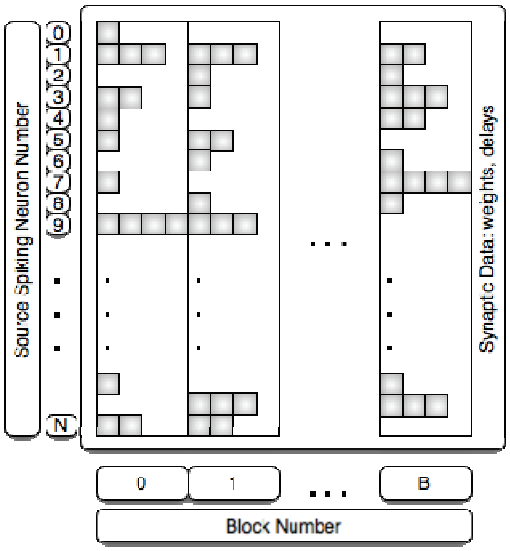


Figure 5.2-3 Synaptic matrix representation

Y coordinate is a spiking source neuron; x coordinate is all target synapses of this neuron distributed per block. If a spike event is produced, the entire target array has to be fetched, which makes it suitable for warp-level texture access: although access is random in both space and time, it exhibits: 1) 2D locality in space facilitated by both block-partitioned synaptic structure and target synapse locality, 2) temporal locality with access pattern that depends on a spiking rate.

At the same time, some space in the matrix is unutilized due to the fact that not all neurons have synapses in every block. Compact matrix representation (coordinate format or CSR) is possible; however it requires the use of pointers and therefore, an additional memory access for fetching them. As a result, efficient synaptic data representation is topology-specific, which, in its turn, is application-specific and can be tuned according to a specific case.

Signal propagation is an important aspect of synaptic transmission due to the fact that delays involved in such a process participate in synaptic plasticity including short term plasticity such as STDP. There are several ways to represent signal propagation as discussed in section 3.1. Since the implemented system is a hybrid system, in which exact spike times are emphasized, utilization of an entire range of floating point number representation for a signal time resolution necessitates implementation of a compact merged signal buffer for synaptic signal storage [2]. Although this alternative requires insertion sort, it is compact and better suited for coalesced memory access. At the same time, the insertion sort exhibits parallelism, which makes it suitable for CUDA realization. Figure 5.2-4 depicts parallel implementation of the insertion sort.

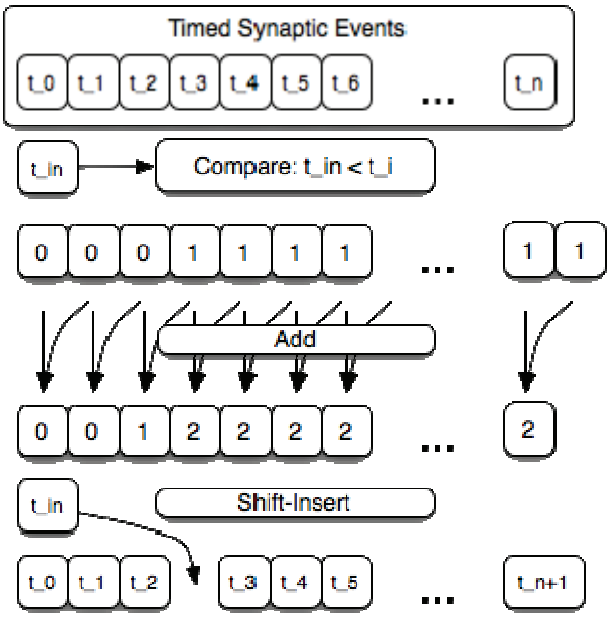


Figure 5.2-4 Insertion of new synaptic event into the queue of events of target neuron

5.3. Interface.

There are various ways to interface the network from outside via state variables of neuron model: current injected in the cell, membrane conductance, spike events and synaptic transmission, ion channel dynamics. Although injected current is an artificial way to interface each neuron and can be construed as a variable current source, it is somewhat comparable to an interaction with another cell via gap junction. Besides, artificially injected current is frequently used in voltage clamp experiments targeted for measuring voltage-current relationship of the cell. Interface via injected current provides a simple translation of continuous variable dynamics to spike events via first layer of interfaced neurons. Thus, the selection of this type of the interface is appropriate. The value of a current is initialized similar to other parameters of the membrane potential model and communicated with every kernel launch.

Output interface is defined by spike events produced by the network (Figure 4.3-2). In this application for visualization purposes spike events are mapped to a motion image displayed on a screen using OpenGL interoperability. Every pixel represents a neuron in a network. When a spike is produced by a neuron, its corresponding pixel is flashed (Figure 5.3-1).

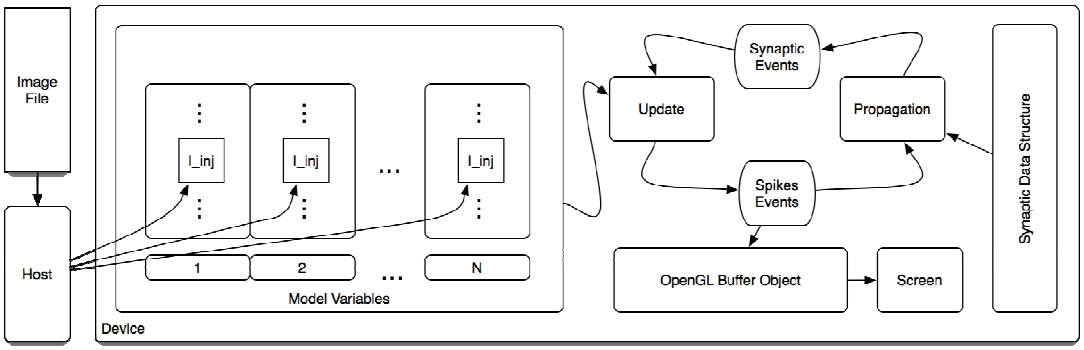


Figure 5.3-1 Interface: block diagram

CUDA allows asynchronous memory operations using page-locked mapped portable memory. This type of memory is a host memory initialized under a unified host-device address space. It has an ability to be read by any host and device thread. Thus, changing current state variable of each neuron can be done by a host thread that reads data from any source (image, video, file etc), translates this data to the injected current value and stores in page-locked mapped portable memory allocated for this purpose. Because this is done asynchronously there are potential read-after-write hazards. At the same time these hazards are insignificant for applications when a change in input is gradual compared to simulation time step.

Although the current interface implementation does not utilize benefits of page-locked mapped portable memory, it can be easily extended with it. Asynchronous read provides advantage of high speed interface, when host and device threads are independent of completion of their memory operations. At the same time it imposes certain constraints on volume of data to be transferred, as well as variability of input data

over time. Full characterization of this interface would require its application, for example, in an artificial visual cognition or a computer vision system.

Chapter 6 Results and Analysis

This chapter describes verification procedures, overall results, and provides the analysis for a fully functional implementation of IZ SNN with PS method on CUDA GPU GTX260 device using single precision floating point.

6.1. Verification

Functionality of implementation was verified with the originally provided reference code [2] for networks of various connectivity densities. Voltage traces from all neurons for all integration steps have been compared between CPU and GPU versions. Small insignificant differences were observed at short simulation times, the magnitude of which depends on the connectivity density: the more connections the more and the bigger the differences (Figure 6.1-1). The sources of the differences are most likely dissimilar compilers, hardware floating point units, and floating point standards between CPU and GPU.

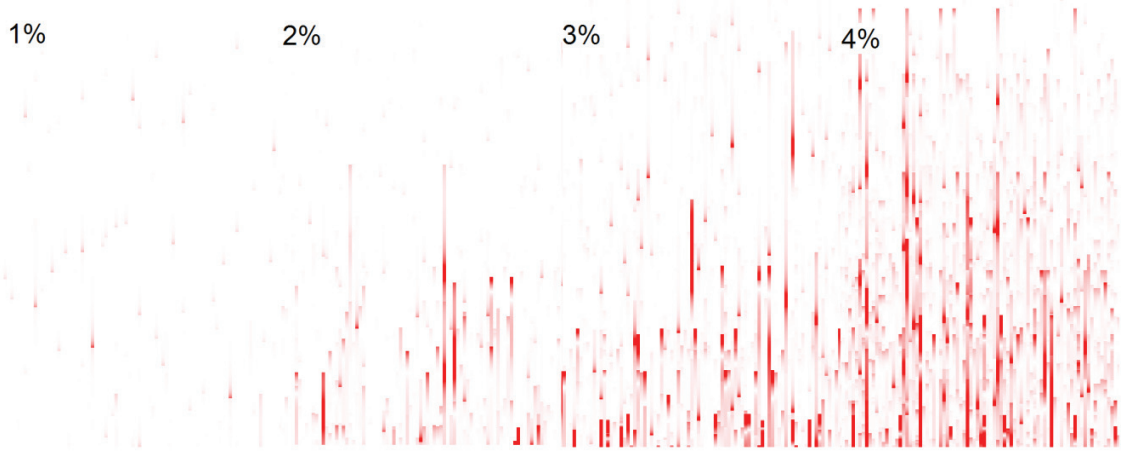


Figure 6.1-1 Raster plot of floating point differences between randomly selected membrane potential traces from SNN executed with reference sequential code (run on Opteron 285, 2.6 GHz) and SNN executed on GPU (GTX260). Measurements taken from 100 ms to 200 ms of simulation time for 4 networks of 768 neurons randomly connected (1%, 2%, 3%, and 4% connectivity). Shading: ■ - difference of 1E-2, - difference of zero. Axes: vertical – time steps (0.25 ms), horizontal – neurons.

With longer simulation time these differences result in time shift of events, which in its turn, causes a spike generation and cancellation. Consequently, due to casualty of events, networks simulated on CPU and GPU diverge completely. For example, in Figure 6.1-2 potential traces of a random neuron start to diverge at about 270th ms of simulation time.

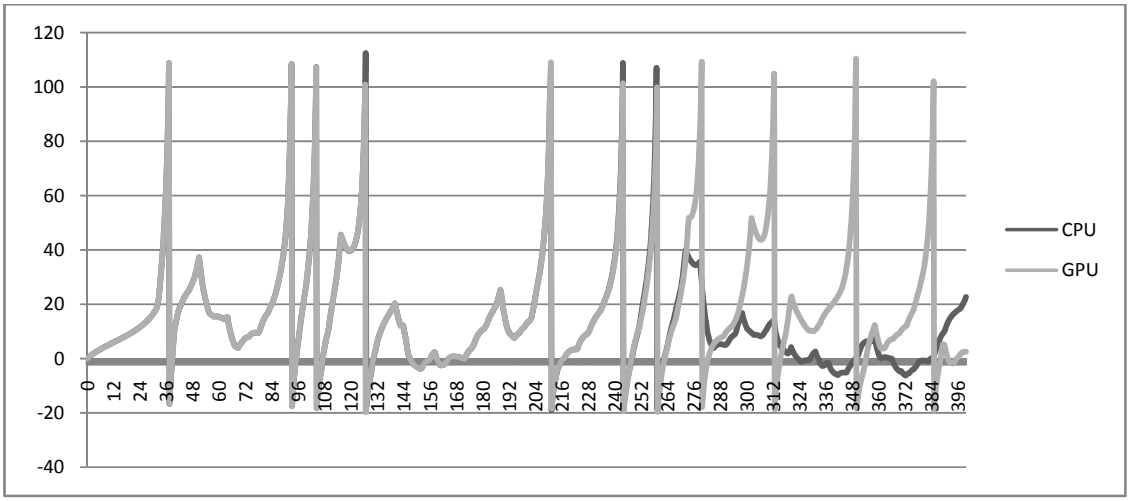


Figure 6.1-2 Divergence of membrane potential traces in CPU vs. GPU implementations. 4% randomly connected network of 3840 neurons is used. Comparison is made between the output of sequential program executed on Opteron 285, 2.6 GHz and that from executed on GTX260 device. Measurements are taken from 0 ms to 400 ms of simulation time. Axes: vertical – membrane potential (mV), horizontal – time (ms).

From Figure 6.1-3 it is evident that the major source of the divergence is a shift of the inhibitory synaptic event from about 272 ms to about 273.5 ms, when membrane potential is higher, and inhibition doesn't prevent further depolarization. However the shifted event still delays the spike. Besides, inhibitory event at about 278.75th ms is missing, which results in higher depolarization followed by a spike and the complete divergence of potential traces. A closer look at the trace of the source neuron of the cancelled event reveals that its cause is a cancellation of an excitatory synaptic event at about 256th ms (Figure 6.1-4).

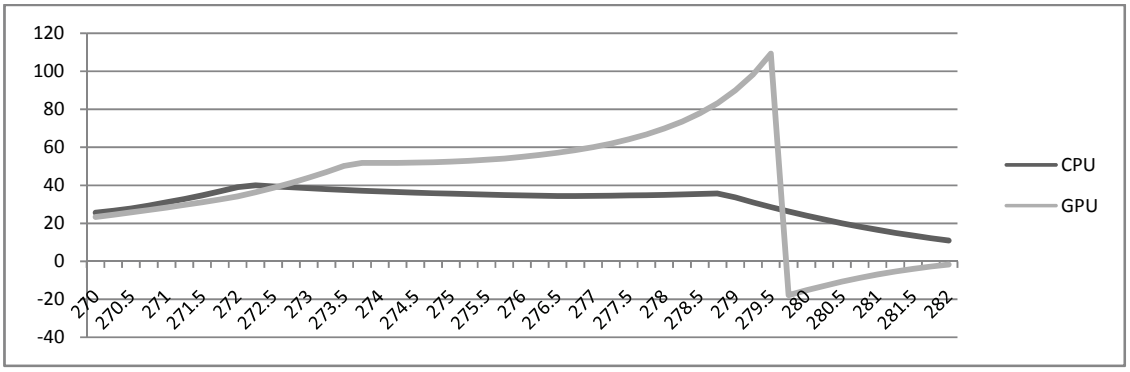


Figure 6.1-3 Divergence of membrane potential traces in CPU vs. GPU implementations. Closer look at the time segment between 270 and 280 ms from Figure 6.1-2. Axes: vertical – membrane potential (mv), horizontal – time (ms).

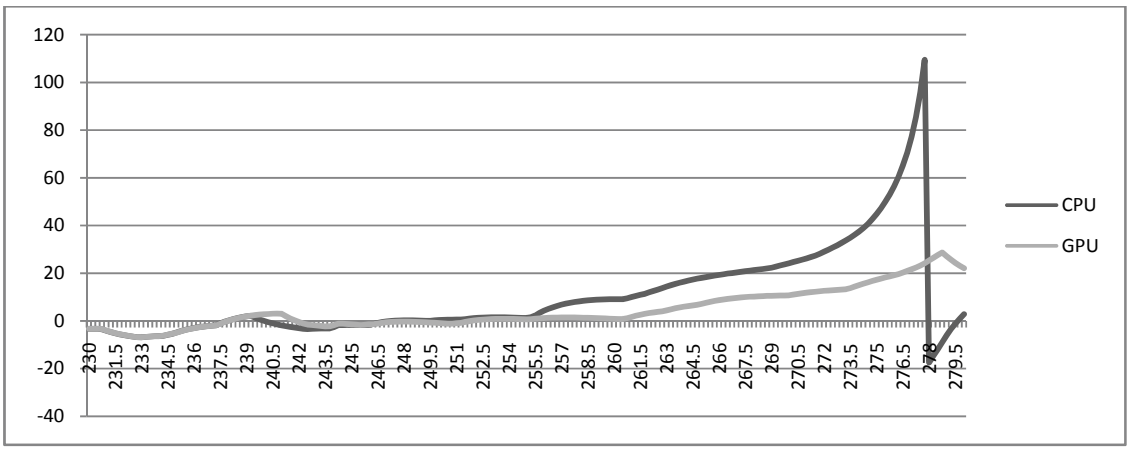


Figure 6.1-4 Spike cancellation in CPU vs. GPU implementation. 4% randomly connected network of 3840 neurons is used. Comparison is made between the output of sequential program executed on Opteron 285, 2.6 GHz and that from executed on GTX260 device. Measurements are taken from 0 ms to 400 ms of simulation time. Axes: vertical – membrane potential (mV), horizontal – time (ms).

Considering casualty of SNN the divergence is expected. Indeed, there is no exact behavior among living biological organisms. Networks are anticipated to diverge either because of their intrinsic architectural differences or because of disturbances from the outside. The important aspect of the divergence caused by a computational system is its adherence to the correctness of fundamental mathematical operations, such as division, multiplication, addition and others, regardless of the data type used (integer, floating

point). If the correctness is confirmed then the next important aspect is the precision: how numbers are truncated, what rounding schemes are used, and others.

The main difficulty is to find a way to eliminate possible program bugs from the list of divergence sources. Hence, another verification procedure is performed: tracking of all synaptic events of the first diverging neuron and comparing them to those of reference simulation. The procedure reveals that divergence originates because of small time differences in the spike and synaptic events between the GPU and the reference CPU implementations. The differences start with small values but become significant as simulation proceeds in time.

The first significantly diverging trace for a neuron from the same network as in the figures above was detected at about 195th ms of simulation time (Figure 6.1-5).

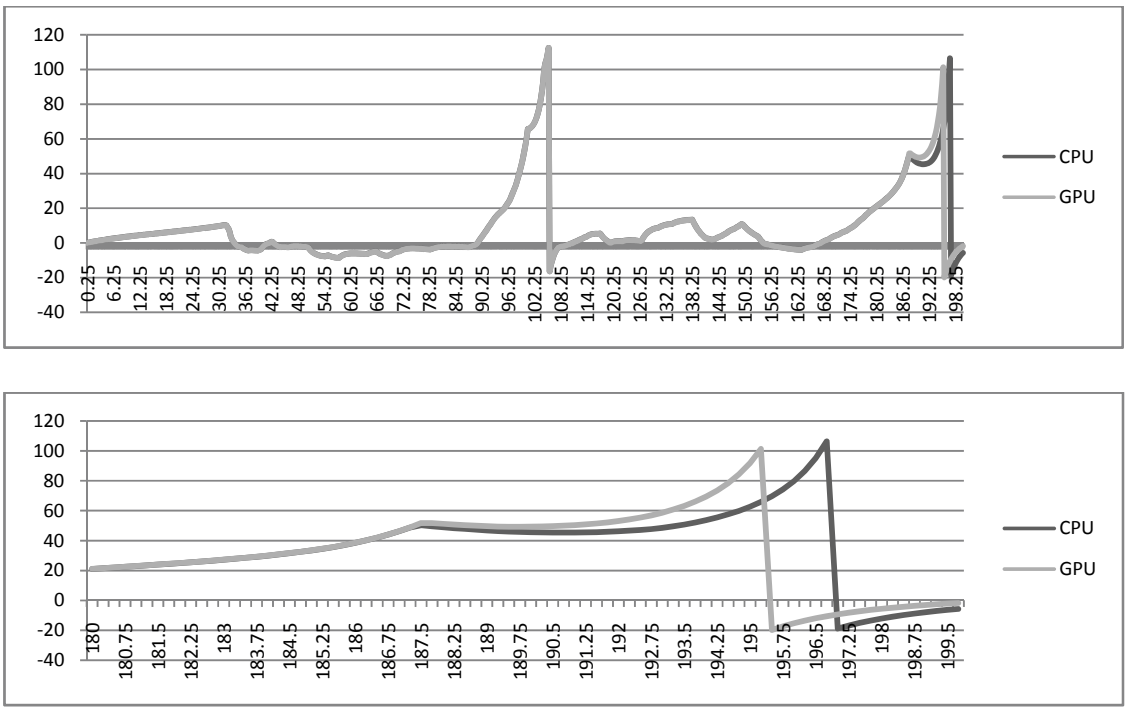


Figure 6.1-5 First significant difference in membrane potential trace in CPU vs. GPU implementations detected at about 195 ms of simulation time. 4% randomly connected network of 3840 neurons is used. Comparison is made between the output of sequential program executed on Opteron 285, 2.6 GHz and that from executed on GTX260 device. Measurements are taken from 0 ms to 200 ms of simulation time. Axes: vertical – membrane potential (mV), horizontal – time (ms).

As it follows from Figure 6.1-5 the cause of the shift in potential traces is a time shift of preceding inhibitory event occurring at about 187.5th ms. Retrieving the potential trace of the source neuron of this event doesn't reveal obvious differences except the fact that the shift occurs after the third spike at about 186.5th ms (Figure 6.1-6). This time is adequate considering the delay of 1ms.

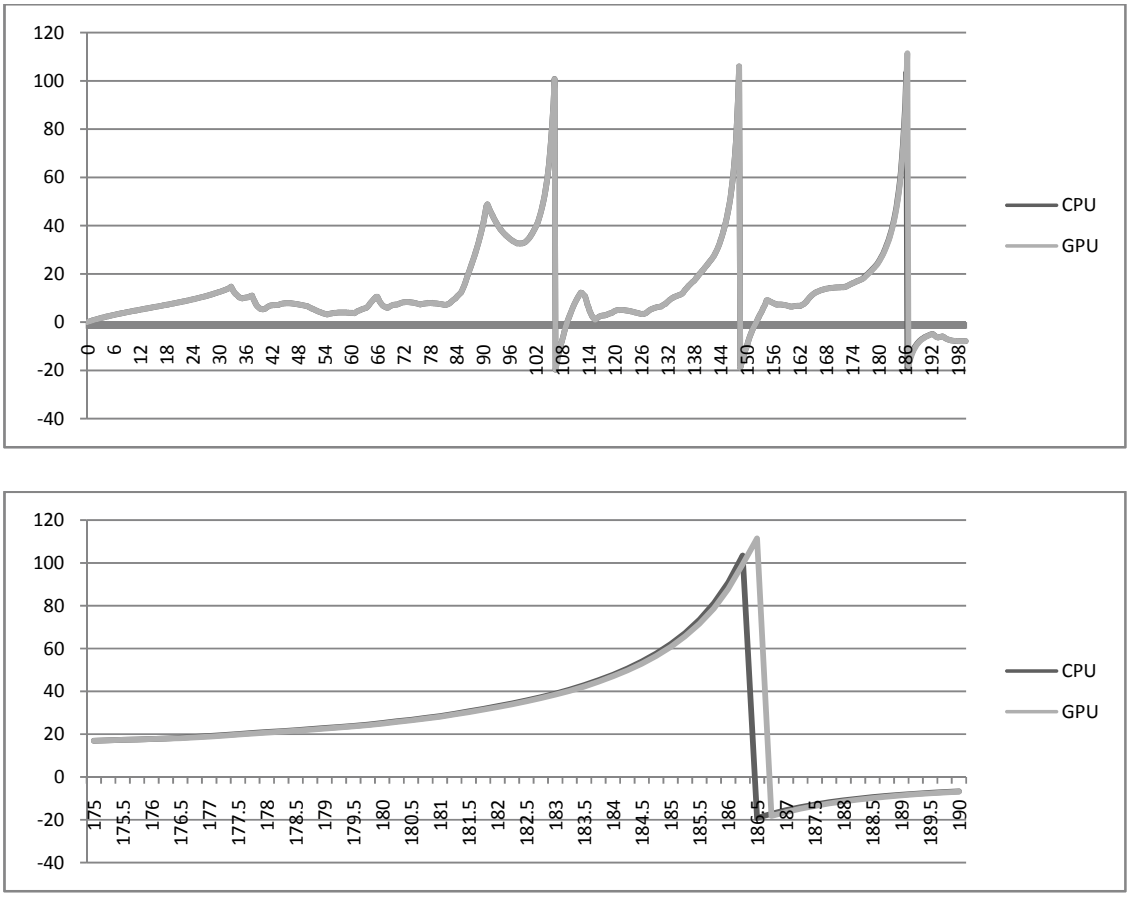


Figure 6.1-6 Potential trace of source neuron for inhibitory event, which causes spike shift in Figure 6.1-5 (CPU vs. GPU implementation). 4% randomly connected network of 3840 neurons is used. Comparison is made between the output of sequential program executed on Opteron 285, 2.6 GHz and that from executed on GTX260 device. Measurements are taken from 0 ms to 200 ms of simulation time. Axes: vertical – membrane potential (mV), horizontal – time (ms).

Further investigation, namely comparison of synaptic event time stamps obtained from CPU and GPU simulations for the neuron producing potential trace depicted in Figure 6.1-6, reveals small differences in time of synaptic events. These differences

increase as simulation proceeds further in time. However, no missing or generated events are found in GPU-based simulation compared to that of CPU-based (Table 6.1-1).

Reference CPU		GPU		Synaptic Weight	GPU-CPU	
Time in integration steps dt	Precise event time	Time in integration steps dt	Precise event time		Delta in dt time	Delta in event time
32.25	32.37959	32.25	32.37959	4.64	0.00	0
32.50	32.72572	32.50	32.72572	-43.43	0.00	0
32.50	32.73192	32.50	32.73192	3.86	0.00	0
33.50	33.54944	33.50	33.54944	3.92	0.00	0
34.50	34.67767	34.50	34.67767	3.91	0.00	0
35.25	35.29513	35.25	35.29512	4.29	0.00	-1E-05
36.50	36.67415	36.50	36.67415	4.42	0.00	0
37.25	37.47728	37.25	37.47728	-43.68	0.00	0
38.50	38.53905	38.50	38.53905	4.65	0.00	0
39.25	39.37458	39.25	39.37458	3.91	0.00	0
39.75	39.96051	39.75	39.96051	4.27	0.00	0
40.50	40.51068	40.50	40.51068	4.66	0.00	0
43.25	43.46223	43.25	43.46224	3.9	0.00	1E-05
54.50	54.57624	54.50	54.57624	3.92	0.00	0
60.75	60.7947	60.75	60.79469	4.63	0.00	-1E-05
63.25	63.43208	63.25	63.43208	3.9	0.00	0
63.75	63.88042	63.75	63.88042	4.67	0.00	0
65.75	65.94031	65.75	65.9403	-47.01	0.00	-1E-05
66.25	66.41985	66.25	66.41984	4.57	0.00	-1E-05
67.00	67.2356	67.00	67.2356	3.93	0.00	0
68.00	68.16737	68.00	68.16736	4.67	0.00	-1E-05
68.25	68.42941	68.25	68.42941	4.37	0.00	0
70.50	70.62214	70.50	70.62213	3.92	0.00	-1E-05
75.75	75.81239	75.75	75.81233	3.91	0.00	-6E-05
81.50	81.53677	81.50	81.53677	4.26	0.00	0
82.25	82.36214	82.25	82.3621	4.57	0.00	-4E-05
83.75	83.77422	83.75	83.77423	4.14	0.00	1E-05
84.75	84.96074	84.75	84.96102	4.63	0.00	0.00028
85.00	85.19913	85.00	85.19913	3.99	0.00	0
85.25	85.26659	85.25	85.26662	3.92	0.00	3E-05
85.75	85.77988	85.75	85.77992	3.94	0.00	4E-05
85.75	85.8244	85.75	85.82437	4.41	0.00	-3E-05
87.00	87.21536	87.00	87.21535	4.4	0.00	-1E-05
88.25	88.49592	88.25	88.49584	4.57	0.00	-8E-05
89.50	89.58337	89.50	89.58338	4.11	0.00	1E-05
89.75	89.84538	89.75	89.84538	3.9	0.00	0
90.75	90.87006	90.75	90.87007	-50.64	0.00	1E-05
94.00	94.16455	94.00	94.16451	3.93	0.00	-4E-05
97.50	97.71494	97.50	97.71494	4.33	0.00	0
99.00	99.14793	99.00	99.14797	3.92	0.00	4E-05
99.50	99.57123	99.50	99.57123	3.99	0.00	0
100.50	100.58944	100.50	100.58943	3.9	0.00	-1E-05
102.25	102.36777	102.25	102.36777	4.41	0.00	0
102.50	102.51791	102.50	102.51789	3.91	0.00	-2E-05
104.00	104.01698	104.00	104.01739	4.11	0.00	0.00041
104.00	104.09686	104.00	104.09677	4.4	0.00	-9E-05
108.00	108.01068	108.00	108.01066	4.14	0.00	-2E-05
108.25	108.30655	108.25	108.30649	4.64	0.00	-6E-05
109.50	109.74025	109.50	109.74019	3.94	0.00	-6E-05

112.25	112.34716	112.25	112.34711	-43.23	0.00	-5E-05
112.25	112.41035	112.25	112.41034	4.53	0.00	-1E-05
113.25	113.32146	113.25	113.32132	-49.28	0.00	-0.00014
114.75	114.84176	114.75	114.84174	3.91	0.00	-2E-05
115.50	115.50854	115.50	115.5085	3.9	0.00	-4E-05
115.50	115.61588	115.50	115.61594	3.92	0.00	6E-05
115.50	115.61927	115.50	115.61921	4.64	0.00	-6E-05
117.75	117.86661	117.75	117.86646	4.54	0.00	-0.00015
119.00	119.22426	119.00	119.22474	4.63	0.00	0.00048
126.25	126.34614	126.25	126.34604	4.23	0.00	-0.0001
127.00	127.06908	127.00	127.06898	4.11	0.00	-0.0001
130.50	130.51711	130.50	130.51697	4.14	0.00	-0.00014
131.50	131.53841	131.50	131.5367	3.92	0.00	-0.00171
135.00	135.11244	135.00	135.11221	4.64	0.00	-0.00023
135.25	135.46214	135.25	135.46199	3.9	0.00	-0.00015
138.00	138.07047	138.00	138.06644	3.91	0.00	-0.00403
142.50	142.58829	142.50	142.58839	4.23	0.00	0.0001
143.50	143.62541	143.50	143.62538	3.94	0.00	-3E-05
149.00	149.19846	149.00	149.19992	4.63	0.00	0.00146
149.75	149.93143	149.75	149.93707	4.14	0.00	0.00564
152.00	152.12854	152.00	152.13705	4.07	0.00	0.00851
153.25	153.41245	153.25	153.41235	4.54	0.00	-0.0001
154.00	154.11479	154.00	154.114	3.86	0.00	-0.00079
154.25	154.37086	154.25	154.37251	4.64	0.00	0.00165
154.50	154.52882	154.50	154.52989	-51.54	0.00	0.00107
156.75	156.75024	156.50	156.74821	3.9	-0.25	-0.00203
160.00	160.17284	160.00	160.16774	3.91	0.00	-0.0051
162.00	162.22157	162.00	162.21153	3.94	0.00	-0.01004
162.75	162.90894	162.75	162.91238	4.23	0.00	0.00344
163.25	163.47241	163.25	163.47324	4.63	0.00	0.00083
172.25	172.3896	172.25	172.39056	4.14	0.00	0.00096
176.25	176.33989	176.50	176.62704	4.54	0.25	0.28715
179.50	179.63051	179.50	179.6102	3.9	0.00	-0.02031
181.00	181.01787	181.00	181.02618	4.01	0.00	0.00831
183.00	183.19052	183.00	183.19119	3.94	0.00	0.00067
186.25	186.38937	186.25	186.38817	-48.59	0.00	-0.0012
192.25	192.31529	192.25	192.31508	-51.54	0.00	-0.00021
193.25	193.41113	193.25	193.42075	4.63	0.00	0.00962
194.50	194.58011	194.50	194.586	-43.23	0.00	0.00589

Table 6.1-1 Event trace table for the neuron with potential trace presented in Figure 6.1-6

Data in Table 6.1-1 shows that some events (156.75 ms and 176.25 ms referenced to CPU time) have to be scheduled at different integration steps in GPU event flow. However, by looking at the delta in event time it is valid to assume that divergence is not originated by the event scheduling algorithm, but rather comes from the part of computation responsible for calculation of even time.

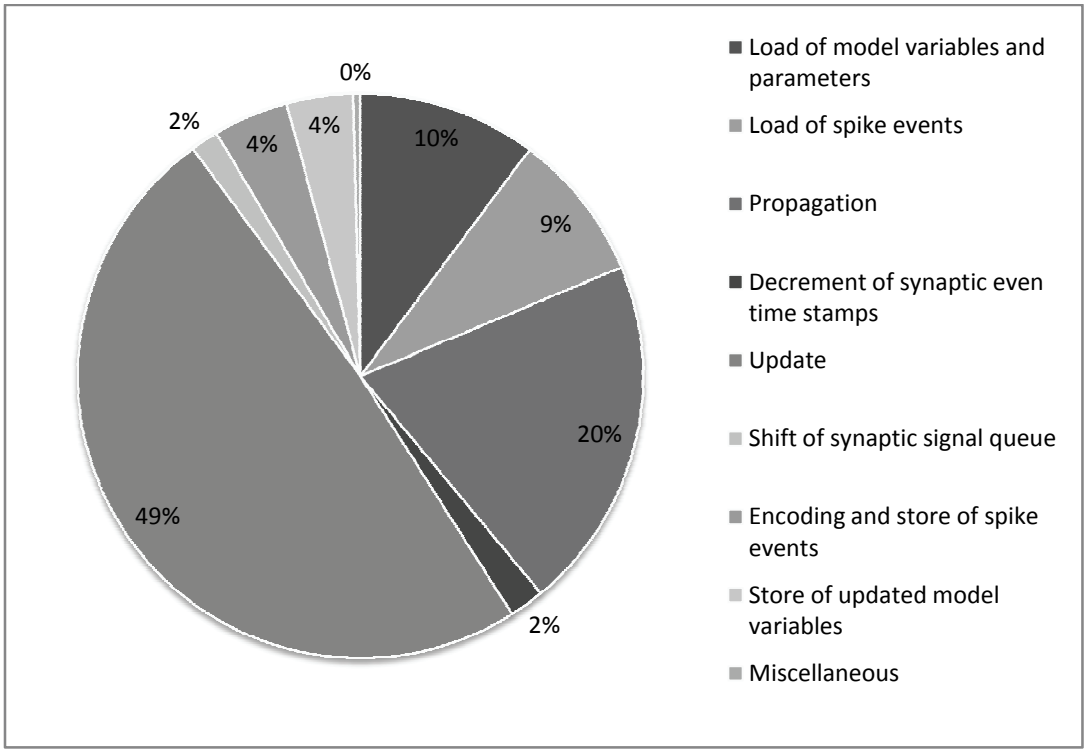
At the same time, the verification performed above for a single potential trace doesn't guarantee that this is true for all diverging neurons. More reliable verification would be based on event comparison within certain tolerance of time and investigation of

causes for a set of first diverging or cancelled/generated events for a range of networks. Such verification incurs some difficulties due to multi-threaded nature of CUDA, but it is not an impossible task if sorting algorithms are applied.

6.2. Execution time

For the profiling purposes a network of 120 blocks x 32 neurons/block = 3840 neurons with equal model variable, constant values, and with equal delay times of 1ms has been simulated on GTX260 device. The integration step is 0.25 ms. The network is 0.5% randomly connected with 80% excitatory and 20% inhibitory synapses, with 73,553 synaptic connections total and with the average of 612 synapses per block. One second of simulation time includes 16,125,709 PS integration steps with maximum PS order of 8 and with error tolerance of 1E-4 for all neuronal state variables. With 32 neurons per block the maximum number of spiked neurons in a block is 3. The network is excited by injecting a current into each cell with a magnitude between 0 pA and 200 pA during first 50 ms of simulation time. The average execution time for this simulation is 978 ms, which is comparable to real time simulation. However, this time is measured starting with the first kernel launch and ending with the last kernel execution. Thus, it doesn't include any other potential overheads, which would be apparent if the system was used in an application. Sequential simulation of the same network on Opteron 285, 2.6GHz processor requires on average 4216 ms.

Chart and Table 6.2-1 demonstrate the distribution of execution time among various parts of a computation for 2% connected network.



Computation	Mean Cycles	Fraction
Update	21,642	0.489
Propagation	8,961	0.203
Load of model variables and parameters	4,522	0.102
Load of spike events	3,780	0.085
Encoding and store of spike events	1,895	0.043
Store of updated model variables	1,673	0.038
Decrement of synaptic even time stamps	885	0.020
Shift of synaptic signal queue	721	0.016
Miscellaneous	134	0.004
Total	44,213	1

Table 6.2-1 Distribution of execution time per computation part for 2% connected network

As it follows from Table 6.2-1, main computational bottlenecks are the update and the propagation phases as well as load operations from global memory. The major source of execution time in the update phase is warp divergence, which results from the adaptive order of the algorithm. The propagation phase has some small sequential parts that can be implemented as parallel algorithms. The load of model variables and parameters can be combined with the load of synaptic event queues in a single operation after rearranging the data structures. Further improvements are discussed in Chapter 7.

All model parameters are neuron specific. Consequently, the size of this data does not allow placing it in constant cache. At the same time, if the assumption that all neurons are the same is valid, then the application of global constants or literals (at compile time) may reduce this part of execution time at the expense of eliminating the feature of neuron differentiation.

6.3. Scalability

Scalability of execution time with network size for a range of network densities is presented in Figure 6.3-1.

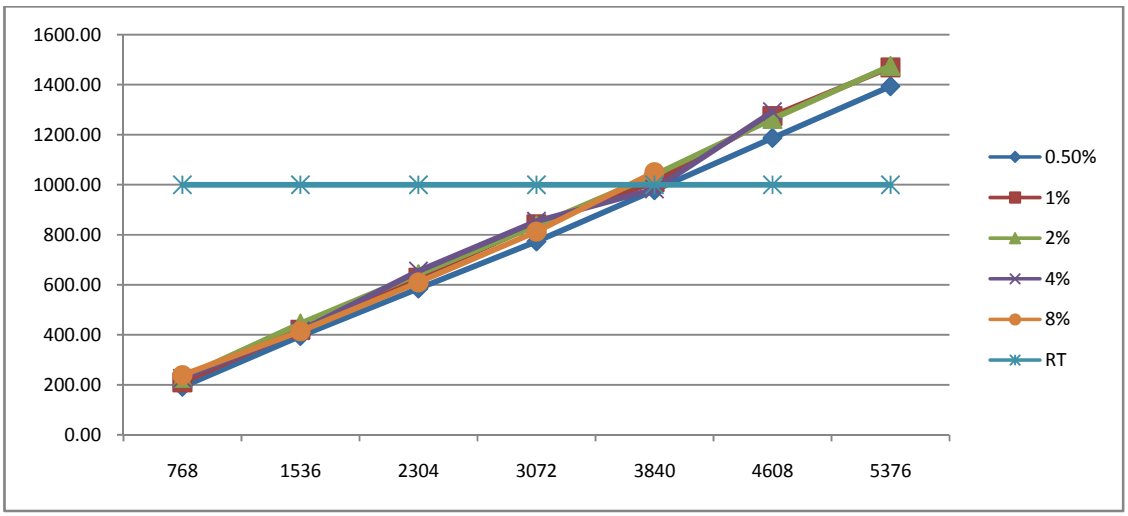


Figure 6.3-1 Scalability of execution time with network size for a range of network densities (0.5% - 8%), 1000 ms of simulated time. Axes: vertical – average execution time (ms), horizontal – network size (number of neurons).

As it can be seen from the figure, the network scales well with the size and introduces a small overhead with network density. For example, in the case of a network with 3840 neurons the density corresponds to the range of 19 synapses per neuron (0.5 % connected) to 307 synapses per neuron (8 % connected) on average. Corresponding execution time varies from 978 ms to 1050 ms. A small size of the overhead results from the size of synaptic queue limited to that of a warp since the algorithm is not optimized for large synaptic queues. In general, linear scalability is due to the fact that there is only one active block per SM. Thus, increasing the number of blocks proportionally to the number of SMs results in linearity.

Due to single kernel implementation, the system is not optimized for large synaptic event queues. Consequently, whole-size queues are fetched to shared memory for the insertion sort, where the memory space is shared among all variables. Dynamic shared memory allocation currently is not implemented for all variables, including synaptic event queues. Thus, for denser networks requiring larger synaptic event queues it is not possible to achieve bigger network sizes compared to sparser networks because of shared memory limitations. For example, 8% connected network is limited to 3840 neurons (Figure 6.3-1). The requirements for the size of synaptic queues and corresponding size of SNN are presented in Figure 6.3-2

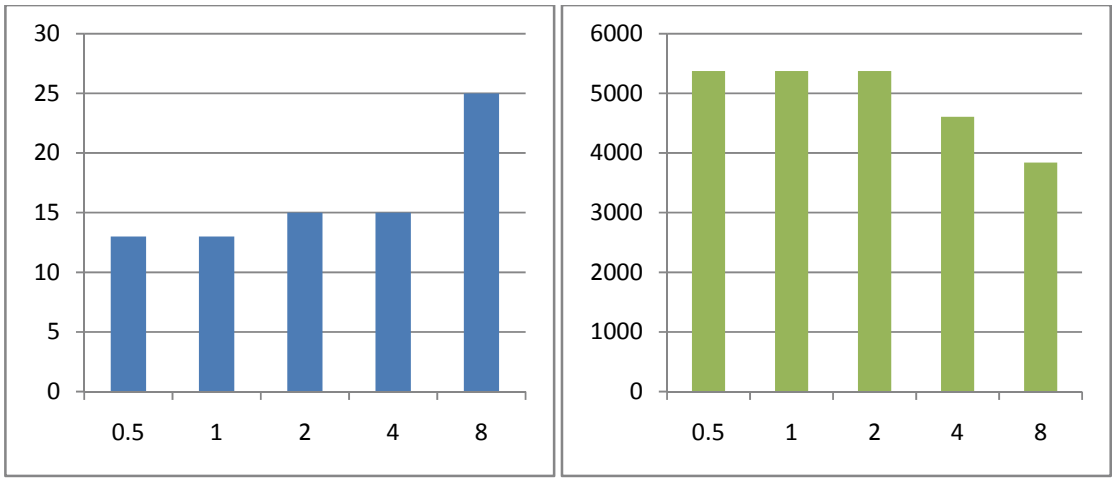


Figure 6.3-2 Left: size of synaptic queue for each neuron vs. % of connectivity (0.5% - 8%). Right: maximum SNN size possible vs. % of connectivity (0.5% - 8%).

Scalability of speedup with network size for various network connectivity densities is presented in Figure 6.3-3.

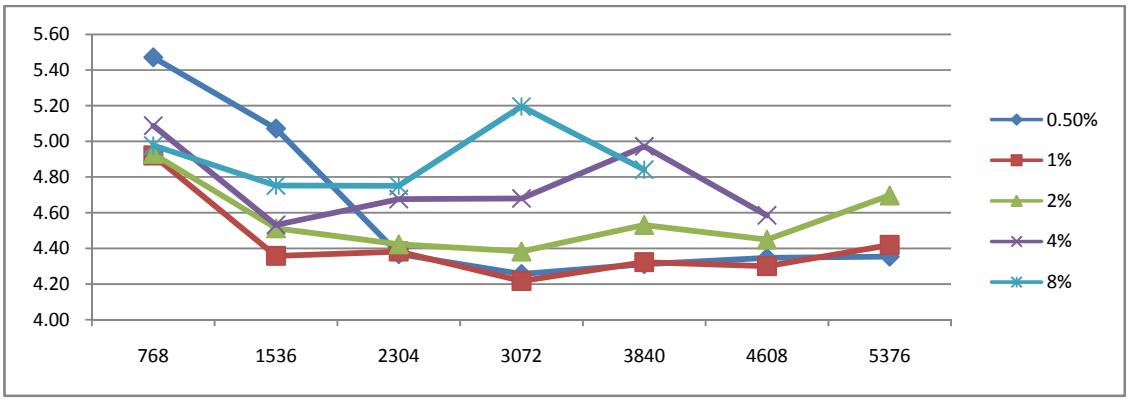


Figure 6.3-3 Scalability of speedup with network size for a range of network densities (0.5% - 8%), 1000 ms of simulated time. Axes: vertical – average speedup, horizontal – network size (number of neurons).

The speedup range is between 4.22 (3072 neurons, 1% connected SNN) and 5.47 (768 neurons, 0.5% connected SNN). At the same time the speedup is an estimated value because CPU execution time has larger standard deviation (up to 373 ms possibly due to the operating system scheduler) compared to that of GPU (up to 1.52 ms).

Figure 6.3-4 demonstrates a linear increase in mean of synaptic connections per block with network size.

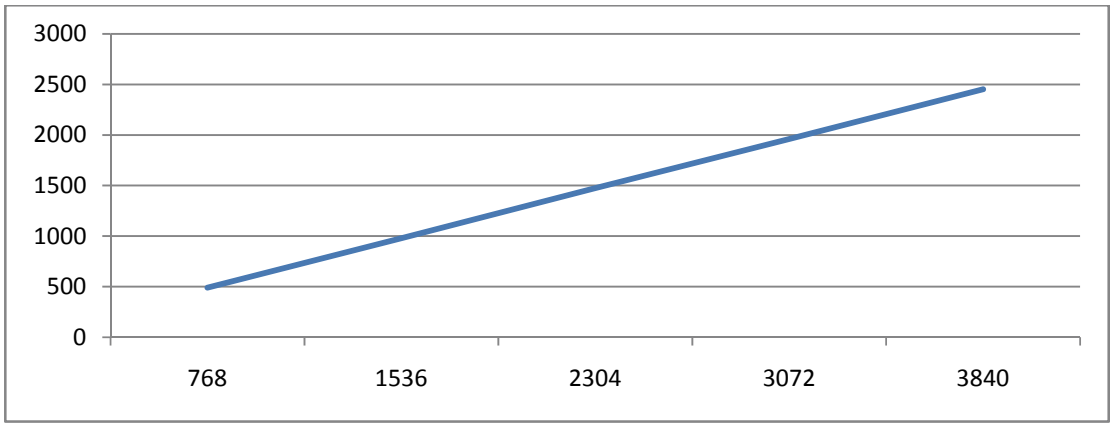


Figure 6.3-4 Mean synaptic connections per block with network size, 2% connected network. Axes: vertical – the number of synaptic connections, horizontal – the network size (number of neurons).

Scalability of execution time fraction for the case of 2% connected network is depicted in Figure 6.3-5.

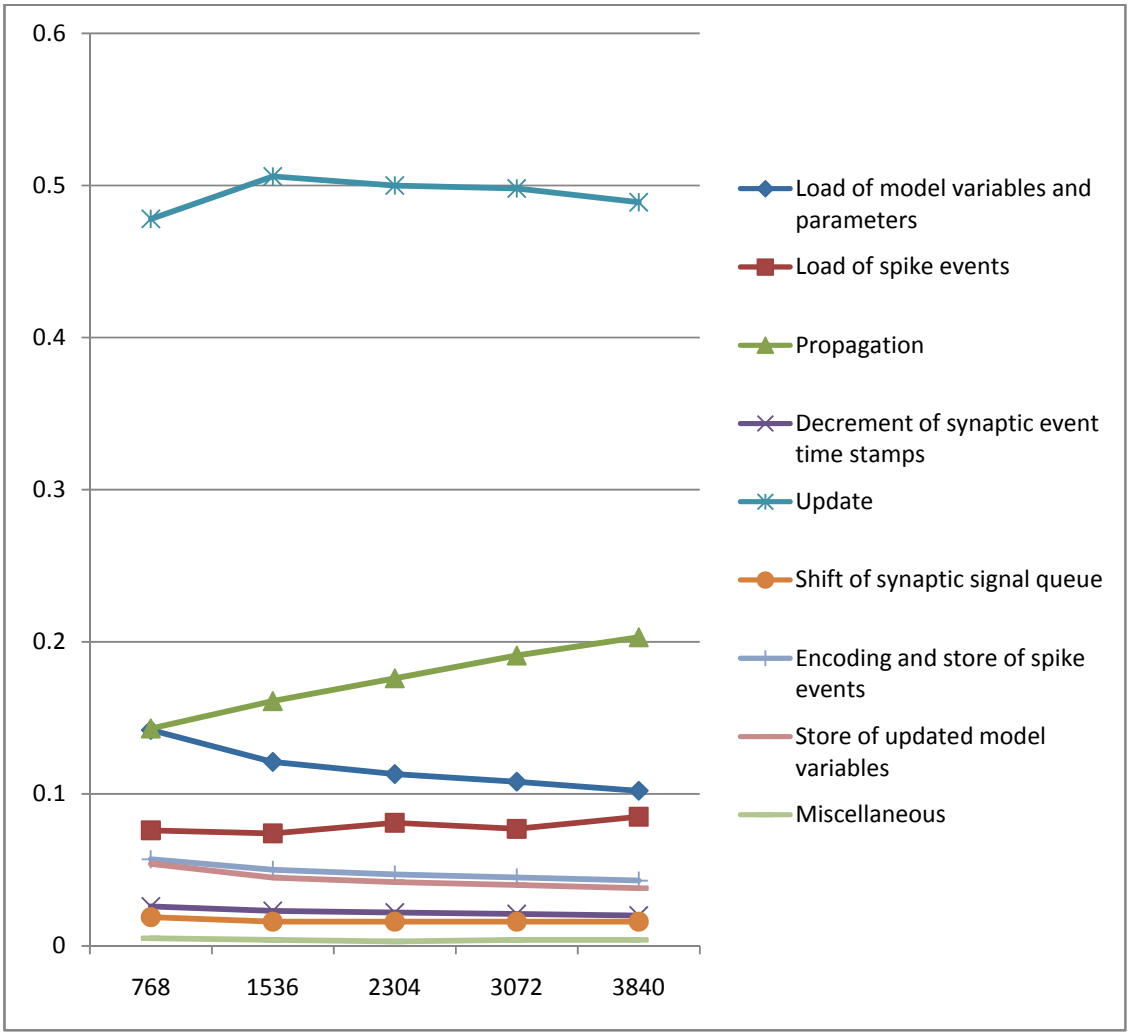


Figure 6.3-5 Scalability of execution time fraction with network size, 2% randomly connected network. Axes: vertical – fraction of execution time, horizontal – network size (number of neurons).

The variation of fractions is insignificant. However, Figure 6.3-5 shows that the fraction of update phase and model variable load yields to propagation and spike event load fractions as a response to a linear increase in the mean value of synaptic connections per block (Figure 6.3-4). This signifies a slight shift in scalability towards the update

phase, since its computation can accommodate larger network at the expense of smaller increase in execution time compared to that of other parts of computation.

A rough estimation of existing CUDA-enabled GPU implementations of SNNs is depicted in

Table 6.3-1. For details the reader is directed to the cited sources. Values in column "Error" are obtained according to the method of error measurement applied in corresponding work. Integration step size is indicated as well. In this work, the error value is the error tolerance, a change in the state variable values between each PS step.

Source	Device	Reference CPU	System	Method	Neuron	Net Size	Speedup	RT	Error
This work	GTX260	Opteron 285 2.6 GHz	Hybrid	PS	IZ	3, 840	4.3-5.7	Yes	1E-4
[55]	GTX280	Intel Core2 Duo (6400) 2.13 GHz	Asynchronous	-	IF	16, 384	1.9-6.4	Yes	-
[55]	GTX280	Intel Core2 Duo (6400) 2.13 GHz	Synchronous	-	IF	81, 920	35	Yes	25%, Δt=5ms
[54]	GTX280	Intel Core2 Duo (6400) 2.13 GHz	Synchronous	Euler	IZ	100K	24	1.5 slower	Δt=1ms

Table 6.3-1 Comparison of existing CUDA GPU implementations

6.4. Interface

OpenGL interoperability provides a network interface and allows visualization of spikes in real time. In Figure 6.4-1 a gray scale value of each pixel in the image is mapped to a current value injected in a corresponding neuron of weakly (0.01%) connected network (lighter shading is mapped to a stronger current).

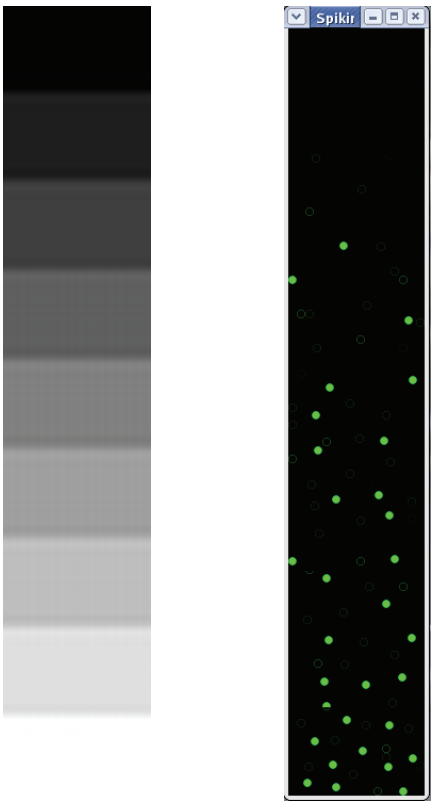


Figure 6.4-1 Spike response with OpenGL interface

As a result, the network reacts with spikes (right image). This basic model of a retina receptor layer can be utilized in larger SNNs for image processing.

Chapter 7 Conclusions and Future Work

Real-time performance for a hybrid system of IZ SNN with 3840 neurons was achieved. The system is implemented based on CUDA-enabled GPU GTX260. Algorithmic speedup is 4.3 referenced to Opteron 285, 2.6 GHz for 0.5% connected network. Speedup range is between 4.22 (3072 neurons, 1% connected SNN) and 5.47 (768 neurons, 0.5% connected SNN).

Implemented as a hybrid type, the system has a potential for applications requiring high accuracy of synaptic event time. This becomes prominent if the system is augmented with STDP rules (see section 2.1.3). Absence of quantization error in hybrid or event-driven systems makes them preferable if high accuracy is needed.

The range of network sizes that the system is capable to run in real time is suited for applications that utilize small and accurate SNNs, such as robotics. Image processing, as a rule, requires mapping every pixel with several layers of neurons, which demands larger SNNs that do not require fine-time handling of synaptic events.

The system provides real-time interface: any neuron can be interfaced with the use of an injected current (model variable) as an input, and spike times as an output. This interface can be used in various applications. For example, in control, robotics, classification, signal processing, and others. The interface can be further improved with page-locked mapped portable memory and dynamic asynchronous memory read-write transactions.

Functionality of the system was verified with reference sequential implementation provided in [2]. Transient membrane potential waveforms for the entire network were compared to those of reference simulation. Small random discrepancies exist within short simulation time (on the order of 1E-2 for 100 ms of simulation time depending on connectivity density), which is expected due to differences in compiler and hardware floating point unit architectures between GPU and CPU. Over a long period of simulation time or for a network with higher degree of connectivity these deviations propagate, which results in a complete output divergence of two networks (sequential CPU version

and parallel GPU version). However, this is expected due to causality of networks. Verification of the first significantly diverging potential trace of a neuron in 4%-connected network with a size of 3840 neurons revealed that there are no cancelled or generated events compared to the reference event flow. However, there are differences in time of events between CPU and GPU event flow. Magnitude of the differences grows as simulation proceeds in time.

At the same time, more optimizations are still required to apply in order for the system to be comparable with existing implementations. As it was shown, within real-time domain the implementation exhibits linear scalability with a network size and relatively small overhead is introduced with increase of a network density (percent of network connectivity). Beyond real time the network size and connectivity density are limited by the size of synaptic queues processed within shared memory boundaries. This bottleneck can be eliminated if multi-kernel implementation is utilized. Considering the fact that density of synaptic connections in animal and in human brain sometimes achieves several thousand per neuron (hundreds of thousand for a Purkinje cell), the importance of reducing the effects of the bottleneck becomes prominent.

Single kernel implementation incurs high register utilization (49 registers for 3840 neurons) in the first place and high shared memory utilization (about 16 KB) in the second place. The resulting effect is 25% of device full occupancy. These limitations dictate all computational parts of the kernel to be within a single boundary of active threads and active blocks. Consequently, limitations prohibit the optimum allocation of each individual part of computation within its own boundary of active threads and active blocks. Multi-kernel implementation is expected to break this bottleneck for some parts of computation. As a consequence, it may provide the optimum thread per block and block per grid partitioning independently for each kernel. Thus, it may maximize device utilization for each task. In addition, if implemented either with page-locked mapped memory or as a multi-stream version, it can support overlapping communication between host and device with computation on the device.

The update part of the computation has to be optimized further in order to accommodate parts of computation that result in warp divergence: the adaptive order of

PS integration step, variability in the quantity of synaptic events per neuron to be processed within the update phase, and probabilistic nature of spiking neurons. Therefore, dynamic scheduling techniques can be utilized: grouping, buffering, producer-consumer models, and others.

Other improvements include, but not limited to:

- Exploiting additional parallelism within propagation phase.
- Parallel implementation of Newton-Raphson method or replacing it with parallel-friendly algorithm.
- Verifying the system response to the device scalability beyond GTX260, enabling multi-GPU functionality.
- Optimization of synaptic data structures and their access.
- Parsing connectivity map from a file.
- Extending the range of biological features: synaptic plasticity, STDP, LTP-LTD.
- Optimization of spike event communication, spike data size reduction.
- Optimization of block-level network allocation, reduction of inter-block connections based on provided topology and heuristics.
- Developing reliable automatic verification system.
- Extending to a double precision floating point.
- Test delays with arbitrary time (has to modify reference code).

Bibliography

[1] R. Brette, et al., "Simulation of networks of spiking neurons: A review of tools and strategies," *Journal of Computational Neurscience*, vol. 23, no. 3, pp. 349-398, 2007.

[2] R. Stewart and W. Bair, "Spiking neural network simulation: numerical integration with the Parker-Sochacki method," *Journal of Computational Neuroscience*, vol. 27, no. 1, pp. 115-33, Aug. 2009.

[3] E. M. Izhikevich, "Simple model of spiking neurons," *Neural Networks, IEEE Transactions on*, vol. 14, pp. 1569--1572, 2003.

[4] E. Kandel, J. Schwartz, and T. Jessell, *Principles of Neural Science*. McGraw-Hill Medical, 2000.

[5] N. Bocquet, et al., "X-ray structure of a pentameric ligand-gated ion channel in an apparently open conformation," *Nature*, vol. 457, pp. 111--114, Jan. 2009.

[6] D. C. Gadsby, "Structural biology: Ion pumps made crystal clear," *Nature*, vol. 450, pp. 957--959, Dec. 2007.

[7] A. Hodgkin and A. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve.," *The Journal of physiology*, vol. 117, pp. 500--544, Aug. 1952.

[8] B. Sakmann and E. Neher, Eds., *Single-Channel Recording*. New York, USA: Plenum, 1985.

[9] P. Dayan and L. F. Abbott, *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press, 2005.

[10] E. M. Izhikevich, "Which model to use for cortical spiking neurons?," *Neural Networks, IEEE Transactions on*, vol. 15, pp. 1063-1070, Sep. 2004.

[11] C. Koch and I. Segev, Eds., *Methods in neuronal modeling: From synapses to networks*. MIT Press, 1989.

[12] P. J. Sjostrom, E. A. Rancz, A. Roth, and M. Hausser, "Dendritic Excitability and Synaptic Plasticity," *Physiol. Rev.*, vol. 88, pp. 769--840, Apr. 2008.

[13] R. C. Malenka and M. F. Bear, "LTP and LTD: An Embarrassment of Riches," *Neuron*, vol. 44, pp. 5--21, Sep. 2004.

[14] Y.-D. Zhou, C. D. Acker, T. I. Netoff, K. Sen, and J. A. White, "Increasing Ca2+ transients by broadening postsynaptic action potentials enhances timing-dependent synaptic depression," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 102, pp. 19121-19125, 2005.

[15] B. M. Kampa, J. Clements, P. Jonas, and G. J. Stuart, "Kinetics of Mg2+ unblock of NMDA receptors: implications for spike-timing dependent synaptic plasticity," *The Journal of Physiology*, vol. 556, pp. 337-345, 2004.

[16] I. P. Pavlov, *Conditioned Reflexes: An Investigation of the Physiological Activity of the Cerebral Cortex*. London, UK: Oxford University Press, 1927.

[17] J.-C. Zhang, P.-M. Lau, and G.-Q. Bi, "Gain in sensitivity and loss in temporal contrast of STDP by dopaminergic modulation at hippocampal synapses," *Proceedings of the National Academy of Sciences*, vol. 106, no. 31, pp. 13028-13033, 2009.

[18] B. Hille, *Ion Channels of Excitable Membranes*. Sinauer Associates, Inc., 2001.

[19] K. M. Hynna and K. Boahen, "Thermodynamically Equivalent Silicon Models of Voltage-Dependent Ion Channels.," *Neural Computation*, vol. 19, pp. 327-350, 2007.

[20] K.-I. Amemori and S. Ishii, "Gaussian Process Approach to Spiking Neurons for Inhomogeneous Poisson Inputs," *Neural Comput.*, vol. 13, pp. 2763--2797, 2001.

[21] L. Lapicque, "Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarisation," *J Physiol Pathol Gen*, no. 9, pp. 620-635, 1907.

[22] W. Gerstner and W. Kistler, *Spiking Neuron Models: An Introduction*. Cambridge University Press, 2002.

[23] E. M. Izhikevich, *Dynamical Systems in Neuroscience: The Geometry of Excitability and Bursting (Computational Neuroscience)*. The MIT Press, 2006.

[24] E. M. Izhikevich and G. M. Edelman, "Large-scale model of mammalian thalamocortical systems," *Proceedings of the National Academy of Sciences*, vol. 105, pp. 3593-3598, 2008.

[25] R. Brette, "Exact Simulation of Integrate-and-Fire Models with Exponential Currents.," *Neural Computation*, vol. 19, pp. 2604-2609, 2007.

[26] W. Singer. (2007, Dec.) Binding by synchrony. Scholarpedia. [Online]. Scholarpedia

[27] E. M. Izhikevich, "Polychronization: Computation with Spikes," *Neural Comput.*, vol. 18, pp. 245--282, 2006.

[28] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Macmillan, 1994.

[29] W. Maas and C. M. Bishop, Eds., *Pulsed Neural Networks*. MIT Press, 1999.

[30] S. M. Bohte and J. N. Kok, "Applications of spiking neural networks," *Inf. Process. Lett.*, vol. 95, pp. 519--520, 2005.

[31] Q. Wu, M. McGinnity, L. Maguire, A. Belatreche, and B. Glackin, "Edge Detection Based on Spiking Neural Network Model," in *Lecture Notes in Computer Science*, 2007, pp. 26--34.

[32] G. Wang and M. Pavel, "A spiking neuron representation of auditory signals," in , vol. 1, 2005, pp. 416-421.

[33] P. Arena, L. Fortuna, M. Frasca, and L. Patane, "Learning Anticipation via Spiking Networks: Application to Navigation Control," *Neural Networks, IEEE Transactions on*, vol. 20, pp. 202-216, Feb. 2009.

[34] M. Gogal, M. Goyal, and C. Watkins, *Computer-Based Numerical & Statistical Techniques (Mathematics)*. Infinity Science Press, 2006.

[35] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, 3rd ed. Cambridge University Press, 2007.

[36] G. E. Parker and J. S. Sochacki, "Implementing the Picard iteration," *Neural, Parallel Sci. Comput.*, vol. 4, pp. 97--112, 1996.

[37] E. Picard, *Traite D'Analyse*. Gauthier-Villars, 1922-1928, vol. 3.

[38] E. A. Coddington and M. Levinson, *Theory of Ordinary Differential Equations*. New York: McGraw-Hill, 1955.

[39] D. E. Knuth, *The art of computer programming*, 3rd ed. Reading, Massachusetts: Addison-Wesley Longman Publishing Co., Inc., 1997, vol. II, Seminumerical Algorithms.

[40] N. Bell and M. Garland, "Efficient Sparse Matrix-Vector Multiplication on CUDA," techreport, 2008.

[41] R. Grimes, D. Kincaid, and D. Young, "ITPACK 2.0 User's Guide. Technical Report CNA-150," Center for Numerical Analysis, University of Texas, 1979.

[42] C. Mead, *Analog VLSI and neural systems*. Addison-Wesley Longman Publishing Co., Inc., 1989.

[43] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128× 128 120 dB 15 μs Latency Asynchronous Temporal Contrast Vision Sensor," *Solid-State Circuits, IEEE Journal of*, vol. 43, pp. 566-576, Feb. 2008.

[44] V. Chan, S.-C. Liu, and A. v. Schaik, "AER EAR: A Matched Silicon Cochlea Pair With Address Event Representation Interface," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 54, pp. 48-59, Jan. 2007.

[45] R. J. Vogelstein, U. Mallik, E. Culurciello, G. Cauwenberghs, and R. Etienne-Cummings, "A Multichip Neuromorphic System for Spike-Based Visual Information Processing," *Neural Comput.*, vol. 19, pp. 2281--2300, 2007.

[46] G. Indiveri, E. Chicca, and R. Douglas, "A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity," *Neural Networks, IEEE Transactions on*, vol. 17, pp. 211-221, Jan. 2006.

[47] A. Ghani, T. M. McGinnity, L. P. Maguire, and J. Harkin, "Area Efficient Architecture for Large Scale Implementation of Biologically Plausible Spiking Neural Networks on Reconfigurable Hardware," in , 2006, pp. 1-2.

[48] L. P. Maguire, et al., "Challenges for large-scale implementations of spiking neural networks on FPGAs," *Neurocomput.*, vol. 71, pp. 13--29, 2007.

[49] E. Ros, E. M. Ortigosa, R. Agis, R. Carrillo, and M. Arnold, "Real-time computing platform for spiking neurons (RT-spike)," *Neural Networks, IEEE Transactions on*, vol. 17, pp. 1050-1063, Jul. 2006.

[50] M. J. Pearson, et al., "Implementing Spiking Neural Networks for Real-Time Signal-Processing and Control Applications: A Model-Validated FPGA Approach," *Neural Networks, IEEE Transactions on*, vol. 18, pp. 1472-1487, Sep. 2007.

[51] A. Cassidy and A. G. Andreou, "Dynamical digital silicon neurons," in , 2008, pp. 289-292.

[52] J. Espenshade, "Scalable Framework for Heterogeneous Clustering of Commoidty FPGAs," Rochester Institute of Technology Thesis, 2009.

[53] *NVIDIA CUDA Programming Guide 2.2*. 2008.

[54] J. Nageswaran, N. Dutt, J. Krichmar, A. Nicolau, and A. Veidenbaum, "A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processorsâ˜†," *Neural Networks*, Jul. 2009.

[55] J. M. Nageswaran, N. Dutt, Y. Wang,, and T. Delbrueck, "Computing spike-based

convolutions on GPUs," in , 2009, pp. 1917-1920.

[56] P. Rhodes. (2009, Jul.) Evolved Machines. [Online]. http://www.evolvedmachines.com/

[57] J. Chakma, "Fusing Neuroscience with Engineering," *BioSynergy*, pp. 24-27, May 2008.

[58] J. Nageswaran, N. Dutt, J. Krichmar, A. Nicolau, and A. Veidenbaum, "A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors," *Neural Networks*, Jul. 2009.

[59] J. M. Nageswaran, N. Dutt, Y. Wang, and T. Delbrueck, "Computing spike-based convolutions on GPUs," in *Intl. Symposium on Circuits And Systems*, 2009, pp. 1917-1920.